OPEN DIGITAL
CONSULTING CO

DRAFT Platform documentation

HOX project

Stuart J Mackintosh

Thursday 27 November 2025

Contents

1 Overview

This chapter introduces the HPS system, its purpose, core architecture, and intended use cases.
It also records the major design decisions that shape the system and defines terminology used throughout. ## Introduction

Stub: Provide a high-level explanation of HPS, its goals, and where it fits into the broader infrastructure platform. # Quick start

A fast path to installing HPS, configuring a cluster, and booting a node.
This section covers the essentials and assumes no prior HPS experience. ## Booting a host

Stub: PXE boot process overview and selecting a host profile. ## First cluster setup

Stub: Use `cluster-configure.sh` to create the first cluster and set its parameters. ## Installation

Stub: Step-by-step guide to deploy `hps-container` with `hps-system` and `hps-config`. ## Prerequisites

Stub: List hardware, OS, packages, network setup, and permissions needed before starting. ## Verification

Stub: Confirming services are active and hosts are provisioned correctly. # Installing HPS

How to install the HPS system on the provisioning node, configure services, and verify readiness. ## Hardware

Depends on the scenario, for example:

- evaluation
- home lab
- small organisation
- production-grade
- maximum performance

1.1 Obtaining HPS

How to acquire HPS source

- HPS Repo link
- Link to ISO downloads

1.2 Dependencies and prerequisites

- Operating system ISO's

1.3 Designing networking and numbering

HPS will manage hosts on a directly connected LAN as it uses lower-level protocols such as AMC address to manage key functions. If HPS is required on another indirectly connected network, then that should have it's own IPN.

When configuring the cluster, make sure to use a network address that doesn;t conflict with anythihg else. HPS should offer ranges that do not conflict with anything that it can detect.

In almost every case, HPS will be implemented on a new network segment.

1.4 Service verification

Stub: Checking that dnsmasq, nginx, supervisord, and other components are running. ## Upgrades and maintenance

Stub: Keeping `hps-system` updated without overwriting configuration files. # Deploying and configuring a cluster with nodes

Creating a cluster, configuring its settings, provisioning nodes, and verifying the environment. ## Cluster configuration

Stub: Setting DHCP interface, storage subnets, OS type, and other cluster settings. ## Cluster creation

Stub: Running `cluster-configure.sh` and choosing cluster parameters. ## Distribution management

Stub: Adding ISOs, extracting PXE trees, and maintaining package repositories. ## Host profiles

Stub: Assigning profiles such as SCH, TCH, DRH, and CCH to nodes. ## Node provisioning

Stub: PXE/iPXE boot workflow and automated node configuration. ## Service management

Stub: Controlling dnsmasq, nginx, supervisord, and other HPS services. ## Verification

Stub: Checking that nodes are deployed correctly and services are operational. # System reference

Static technical reference information for HPS, including function documentation, troubleshooting guides, and configuration details. ## Keysafe Token Authentication Flow

Authorisation sequence

Figure 1.1: Authorisation sequence

1.4.1 Overview

The HPS keysafe system implements a secure, token-based authentication mechanism for establishing rsync sessions between cluster nodes and disaster recovery hosts (DRH). This flow ensures that only authorized nodes can initiate data replication by requiring them to first obtain a cryptographic biscuit token from the Initial Provisioning System (IPS). The IPS validates the requesting node's MAC address against its registry before issuing a time-limited, single-use biscuit token that expires after 60 seconds. This token is then presented to the DRH wrapper, which validates it with the IPS keysafe service before spawning the actual rsync session, creating a secure chain of trust that prevents unauthorized access while enabling efficient data synchronization across the HPS infrastructure.

1.4.2 Flow Diagram

1.4.3 Key Security Features

1. **Biscuit Token Format**: Uses cryptographic biscuit tokens for secure authorization
2. **MAC Address Validation**: IPS validates the source MAC against a pre-registered registry
3. **Time-Limited Tokens**: Biscuit tokens expire after 60 seconds
4. **Single-Use Tokens**: Each token can only be used once (consumed on validation)
5. **Centralized Validation**: All token validation goes through IPS keysafe

1.4.4 Implementation Notes

- The Client Wrapper and DRH Wrapper handle the secure token exchange
- The actual data transfer happens via rsync after authentication
- Token consumption prevents replay attacks
- Short TTL (60 seconds) limits exposure window # Biscuit Token Management Library

Self-contained bash wrapper functions for managing Biscuit authentication tokens in Docker containers.

1.5 Files

- `n_lib_biscuit.sh` - Self-contained library with all biscuit wrapper functions
- `test_n_lib_biscuit.sh` - Comprehensive test suite
- `docker-compose.yml` - Docker compose configuration for biscuit container
- `Dockerfile` - Container build configuration

1.6 Prerequisites

- Docker
- Running biscuit container

No external dependencies required - the library includes built-in file-based storage.

1.7 Storage

Configuration is stored in `/tmp/biscuit_config/` by default. You can override this by setting the `BISCUIT_CONFIG_DIR` environment variable:

```
export BISCUIT_CONFIG_DIR="/path/to/your/config"
```

Stored keys: - `biscuit_private_key` - Private key - `biscuit_public_key` - Public key - `biscuit_ds_token` - Current token

1.8 Container Setup

1. Build and start the biscuit container:

```
docker-compose up -d --build
```

2. Verify container is running:

```
docker ps | grep biscuit
```

1.9 Functions Available

All functions are prefixed with `n_` and use the `biscuit` container by default.

1.9.1 Container Management

- `n_verify_container [container_name]` - Verify container is running

1.9.2 Keypair Management

- `n_keypair_generate [container_name]` - Generate and store keypair
- `n_keypair_get_public` - Retrieve public key
- `n_keypair_get_private` - Retrieve private key

1.9.3 Token Management

- `n_token_generate [container_name]` - Generate 10-second token
- `n_token_get` - Retrieve stored token
- `n_token_inspect <token> [container_name]` - View token contents

- `n_token_verify <token> <public_key> [container_name]` - Verify token
- `n_token_attenuate <token> <datalog_check> [container_name]` - Add restrictions
- `n_token_seal <token> [container_name]` - Seal token (make final)

1.10 Storage

All data stored in cluster_config: - `biscuit_private_key` - Cluster private key - `biscuit_public_key` - Cluster public key - `biscuit_ds_token` - Current token

1.11 Usage Examples

1.11.1 Generate a keypair

```
source n_lib_biscuit.sh
n_keypair_generate
```

1.11.2 Generate a token

```
token=$(n_token_generate)
echo "Generated token: $token"
```

1.11.3 Verify a token

```
public_key=$(n_keypair_get_public)
n_token_verify "$token" "$public_key"
```

1.11.4 Attenuate a token with restrictions

```
restricted_token=$(n_token_attenuate "$token" 'check if
 ↪  operation("read");')
```

1.11.5 Seal a token

```
sealed_token=$(n_token_seal "$token")
```

1.12 Running Tests

Simply ensure the biscuit container is running and execute:

```
bash test_n_lib_biscuit.sh
```

The test suite includes: - Container verification - Keypair generation (with overwrite protection) - Key retrieval - Token generation and storage - Token inspection - Token verification - Token attenuation - Token sealing - Token expiration (10 second TTL test)

1.13 Notes

- Tokens have a 10-second TTL (hardcoded)
- Keypair generation prompts for confirmation if keypair already exists
- All functions use `hps_log` for error/info logging
- Sealed tokens cannot be attenuated further
- Container name defaults to “biscuit” but can be overridden

1.14 Integration with HPS

When integrating with your HPS system, you can easily adapt the storage backend:

1. Replace the `biscuit_config_*` functions to use your `cluster_config` or `host_config`
2. The rest of the library will work unchanged

Example adaptation:

```
biscuit_config_set() {
  cluster_config "set" "$1" "$2"
}

biscuit_config_get() {
  cluster_config "get" "$1"
}

biscuit_config_exists() {
  cluster_config "exists" "$1"
}
```

1.15 Cluster management

Clusters are managed by OpenSVC.

The central config file is generated on the IPS and downloaded on demand by cluster hosts. It is dynamically built based on the cluster config.

1.15.1 OpenSVC

we are using v3

Note:

Docs are incomplete.

- V3 docs: https://book.opensvc.com/a
- V2 docs: https://docs.opensvc.com/latest/

Thing you want to set	Where / How	
Agent log path/level	`opensvc.conf` (`log_file`, `log_level`)	
Agent TCP listener / Web UI ports	`opensvc.conf` (`listener_port`, `web_ui*`)	
Node local tags for default behavior	`opensvc.conf` (`tags`)	
Cluster members / node IPs / names	`cluster.conf`	
Service resources (zpool/zvol/f-s/ip/share)	`om ... create/set` → lives under `services/*`	
Placement rules (tags=storage, nodename=…)	`om mysvc set --kw placement=…` (in service cfg)	
Start/stop/provision services	‘om mysvc start	stopprovision‘
Distribute service configs to other nodes	`om mysvc push` / `om mysvc sync`	

1.15.1.1 Useful commands:

= the defined service name

om print config Prints the config for the service
om config validate Checks the config for the node and reports on errors
om purge purge, unprovision and delete are asynchoronous and do things on all node with a object instance

1.15.1.2 References

1.16 Environment variables

Stub: Explanation of exported variables and their purpose. ## Library functions

There are two main libraries of functions, the hps functions, and host functions.

hps functions are used during the cluster build and configure process whereas the host functions are available on the running host.

1.17 OpenSVC V3 Alpha Cheatsheet

Version tested: v3.0.0-alpha87

1.17.1 Basic Object Management

```
# Check version
om -v

# Monitor cluster status
om mon

# List nodes
om node ls

# List services
om svc ls

# List all objects
om all ls
```

1.17.2 Service Management

1.17.2.1 Create Service

```
# Basic creation
om <service-name> create

# With keywords
om <service-name> create --kw <section>.<key>=<value>

# Example
om mysvc create --kw task#hello.type=host --kw
 ↪  task#hello.command="echo hello"
```

1.17.2.2 Service Operations

```
# View configuration
om <service-name> config show

# Delete service
om <service-name> delete

# View logs
om <service-name> logs

# View status
om <service-name> print status
```

1.17.3 Task Resources

1.17.3.1 Create Task

```
# Basic task
om mysvc create \
  --kw task#name.type=host \
  --kw task#name.command="<command>"

# Source bash functions and execute
om mysvc create \
  --kw task#name.type=host \
  --kw task#name.command=". /path/to/functions.sh && my_function"
```

1.17.3.2 Run Tasks

```
# Run specific task
om <service-name> run --rid task#name

# Run on all nodes
om <service-name> run --rid task#name --node=\*

# Get session ID for tracking
# Output shows: OBJECT NODE SID
```

1.17.3.3 View Task Output

```
# View logs with session filter
om <service-name> log --filter SID=<session-id>

# Or use journalctl (local only)
journalctl SID=<session-id>
```

1.17.4 Environment Variables Available in Tasks

Tasks automatically receive these environment variables:

```
OPENSVC_ACTION=run
OPENSVC_NAME=<service-name>
OPENSVC_SVCNAME=<service-name>
OPENSVC_KIND=svc
OPENSVC_SID=<session-id>
OPENSVC_ID=<object-id>
OPENSVC_LEADER=0|1
OPENSVC_SVCPATH=<service-path>
OPENSVC_RID=task#<name>
OPENSVC_NAMESPACE=root
```

1.17.5 Sync Resources

1.17.5.1 Create Sync Resource

```
# Rsync between nodes
om mysvc create \
  --kw sync#name.type=rsync \
  --kw sync#name.src="/source/path" \
  --kw sync#name.dst="/dest/path"

# Provision sync
om mysvc provision --rid sync#name
```

Note: Sync resources distribute files FROM the node running the service TO other nodes. Single-node setups will show “no target nodes”.

1.17.6 Configmaps

1.17.6.1 Create and Manage Configmaps

```
# Create configmap (note: cfg/ prefix required)
om cfg/name create

# List configmaps
om cfg ls

# Add key to configmap
om cfg/name key add --name=filename.ext --value='content'

# Add key from file
om cfg/name key add --name=filename.ext --from=/path/to/file

# List keys
om cfg/name key list
```

```
# View key content
om cfg/name key decode --name=filename.ext

# Delete configmap
om cfg/name delete
```

1.17.6.2 Configmap Limitations in V3 Alpha

- The `configs` parameter in tasks does NOT currently expose configmap data as environment variables or files
- Configmap data is stored base64-encoded in `/etc/opensvc/cfg/<name>.conf` under `[data]` section
- **Workaround:** Use direct file paths or inline functions instead

1.17.7 Naming Conventions

1.17.7.1 Valid Names

- Use hyphens (-), not underscores (_)
- Lowercase only
- Must comply with RFC952 (DNS naming rules)
- Examples: `my-service`, `test-functions`, `storage-mgmt`

1.17.7.2 Object Path Formats

- Services: `<name>` (no prefix)
- Configmaps: `cfg/<name>`
- Secrets: `sec/<name>` (assumed, not tested)
- Volumes: `vol/<name>` (assumed, not tested)

1.17.8 Resource Section Naming

```
# Format: <type>#<name>
task#hello
sync#files
app#webserver
disk#data
```

1.18 Working Patterns

1.18.1 Pattern 1: Simple Task Execution

```
om test create \
  --kw task#hello.type=host \
  --kw task#hello.command="echo 'Hello from task'"
```

```
om test run --rid task#hello
```

1.18.2 Pattern 2: Tasks with Bash Functions

```
# Create functions file on node(s)
cat > /opt/opensvc/functions.sh << 'EOF'
#!/bin/bash
my_function() {
    echo "Output from function"
}
EOF

# Create service
om mysvc create \
  --kw task#run.type=host \
  --kw task#run.command=". /opt/opensvc/functions.sh &&
   ↪  my_function"

om mysvc run --rid task#run
```

1.18.3 Pattern 3: Multi-Node Execution

```
# Create service with multiple nodes
om cluster-task create \
  --kw nodes="node1,node2,node3" \
  --kw task#check.type=host \
  --kw task#check.command="hostname; df -h"

# Run on all nodes
om cluster-task run --rid task#check --node=\*

# Returns SIDs for each node
# View specific node output
om cluster-task log --filter SID=<session-id>
```

1.18.4 Pattern 4: File Distribution (When Multiple Nodes Exist)

```
# Create sync resource
om distribute create \
  --kw nodes="node1,node2" \
  --kw sync#files.type=rsync \
  --kw sync#files.src="/local/file.sh" \
  --kw sync#files.dst="/remote/file.sh"

# Provision to distribute
om distribute provision --rid sync#files
```

```
# Then use in tasks
om distribute create --kw task#run.type=host \
  --kw task#run.command=". /remote/file.sh && function_name"
```

1.19 Known Limitations (V3 Alpha)

1. **Configmap Exposure:** `configs` parameter doesn't expose data to tasks
2. **Single Node Sync:** Sync resources need multiple nodes to function
3. **No Service Prefix:** Services use bare names, not `svc/<name>`
4. **Incomplete Documentation:** Many features undocumented in alpha

1.20 Help Commands

```
# General help
om --help

# Subsystem help
om svc --help
om cfg --help
om node --help

# Command-specific help
om svc create --help
om cfg key add --help
```

1.21 File Locations

```
# Service configs
/etc/opensvc/<service-name>.conf

# Configmap configs
/etc/opensvc/cfg/<name>.conf

# Service runtime data
/var/lib/opensvc/svc/<service-name>/

# Cluster config
/etc/opensvc/cluster.conf
/etc/opensvc/node.conf
```

Note: This cheatsheet is based on V3 alpha87 testing. Features and syntax may change before GA release. ## Paths

1.21.1 hps provisioning node

1.21.2 hps operating node (TCN, CCN etc)

Storage under /srv/storage

scripts under /srv/scripts

/srv is iscsi mounted

everything else is transient. ## Reference configurations

Stub: Example `cluster.conf`, `host.conf`, and service configuration files. # lib_sozu.sh - Sozu Proxy Management Library

Bash function library for managing Sozu reverse proxy via Docker.

1.22 Overview

This library provides functions prefixed with `s_` for managing Sozu proxy clusters, backends, listeners, and frontends. All functions interact with Sozu running in a Docker container via `docker exec`.

1.23 Prerequisites

- Docker installed
- Sozu container running (default name: `sozu`)
- Sozu config file at `/etc/sozu/config.toml` inside container
- Run commands as root or with appropriate Docker permissions

1.24 Installation

```
source /path/to/lib_sozu.sh
```

1.25 Function Reference

1.25.1 Cluster Operations

1.25.1.1 s_cluster_add

Add a new cluster to Sozu proxy.

Usage:

```
s_cluster_add <cluster_id> <load_balancing_policy>
 ↪  [container_name]
```

Parameters: - `cluster_id` - Unique identifier for the cluster - `load_balancing_policy` - Load balancing policy (e.g., `random`, `roundrobin`) - `container_name` - (Optional) Name of Sozu container (default: `sozu`)

Returns: - `0` on success - `1` if container not found or not running - `2` if sozu command failed

Example:

```
s_cluster_add "web-cluster" "random"
s_cluster_add "api-cluster" "roundrobin" "sozu-prod"
```

1.25.1.2 s_cluster_remove

Remove a cluster from Sozu proxy.

Usage:

```
s_cluster_remove <cluster_id> [container_name]
```

Example:

```
s_cluster_remove "web-cluster"
s_cluster_remove "web-cluster" "sozu-prod"
```

1.25.1.3 s_cluster_list

List all clusters in Sozu proxy.

Usage:

```
s_cluster_list [container_name]
```

Example:

```
s_cluster_list
s_cluster_list "sozu-prod"
```

1.25.2 Backend Operations

1.25.2.1 s_backend_add

Add a backend to a Sozu cluster.

Usage:

```
s_backend_add <cluster_id> <backend_id> <address> [container_name]
```

Parameters: - `cluster_id` - Cluster identifier to add backend to - `backend_id` - Unique identifier for the backend - `address` - Backend address in format `IP:PORT` (e.g., `127.0.0.1:8080`) - `container_name` - (Optional) Name of Sozu container (default: `sozu`)

Returns: - `0` on success - `1` if container not found or not running - `2` if sozu command failed

Example:

```
s_backend_add "web-cluster" "backend-1" "127.0.0.1:8080"
s_backend_add "web-cluster" "backend-2" "192.168.1.10:8080"
 ↪  "sozu-prod"
```

1.25.2.2 s_backend_remove

Remove a backend from a Sozu cluster.

Usage:

```
s_backend_remove <cluster_id> <backend_id> <address>
 ↪  [container_name]
```

Example:

```
s_backend_remove "web-cluster" "backend-1" "127.0.0.1:8080"
s_backend_remove "web-cluster" "backend-1" "127.0.0.1:8080"
 ↪  "sozu-prod"
```

Note: There is no `s_backend_list` function as Sozu does not provide a backend listing command.

1.25.3 Listener Operations

1.25.3.1 s_listener_http_add

Add an HTTP listener to Sozu proxy.

Usage:

```
s_listener_http_add <bind_address> [container_name]
```

Parameters: - `bind_address` - Address to bind listener to in format `IP:PORT` (e.g., `0.0.0.0:8080`) - `container_name` - (Optional) Name of Sozu container (default: `sozu`)

Behaviour: - Automatically adds `--expect-proxy` flag

Returns: - `0` on success - `1` if container not found or not running - `2` if sozu command failed

Example:

```
s_listener_http_add "0.0.0.0:8080"
s_listener_http_add "0.0.0.0:9000" "sozu-prod"
```

Note: By default, Sozu includes HTTP listener on port 80 and HTTPS on port 443.

1.25.3.2 s_listener_http_remove

Remove an HTTP listener from Sozu proxy.

Usage:

```
s_listener_http_remove <bind_address> [container_name]
```

Example:

```
s_listener_http_remove "0.0.0.0:8080"
s_listener_http_remove "0.0.0.0:8080" "sozu-prod"
```

1.25.3.3 s_listener_list

List all listeners in Sozu proxy.

Usage:

```
s_listener_list [container_name]
```

Example:

```
s_listener_list
s_listener_list "sozu-prod"
```

1.25.4 Frontend Operations

1.25.4.1 s_frontend_http_add

Add an HTTP frontend to Sozu proxy.

Usage:

```
s_frontend_http_add <bind_address> <hostname> <cluster_id>
 ↪  [container_name]
```

Parameters: - `bind_address` - Address to bind frontend to in format `IP:PORT` (e.g., `0.0.0.0:80`) - `hostname` - Hostname for the frontend (e.g., `example.com`) - `cluster_id` - Cluster identifier to route traffic to - `container_name` - (Optional) Name of Sozu container (default: `sozu`)

Returns: - `0` on success - `1` if container not found or not running - `2` if sozu command failed

Example:

```
s_frontend_http_add "0.0.0.0:80" "example.com" "web-cluster"
s_frontend_http_add "0.0.0.0:80" "api.example.com" "api-cluster"
 ↪  "sozu-prod"
```

1.25.4.2 s_frontend_http_remove

Remove an HTTP frontend from Sozu proxy.

Usage:

```
s_frontend_http_remove <bind_address> <hostname> <cluster_id>
 ↪  [container_name]
```

Example:

```
s_frontend_http_remove "0.0.0.0:80" "example.com" "web-cluster"
s_frontend_http_remove "0.0.0.0:80" "example.com" "web-cluster"
 ↪  "sozu-prod"
```

1.25.4.3 s_frontend_list

List all frontends in Sozu proxy.

Usage:

```
s_frontend_list [container_name]
```

Example:

```
s_frontend_list
s_frontend_list "sozu-prod"
```

1.25.5 Validation Operations

1.25.5.1 s_validate_endpoint

Validate a Sozu endpoint by making an HTTP request.

Usage:

```
s_validate_endpoint <hostname> <port> [container_name]
```

Parameters: - `hostname` - Hostname to test (will be sent as Host header) - `port` - Port to connect to - `container_name` - (Optional) Name of Sozu container (default: `sozu`)

Behaviour: - Uses curl to make HTTP request to `127.0.0.1:<port>` with Host header - Returns response from backend service

Returns: - `0` on success (HTTP 200-399) - `1` if curl command failed - `2` if HTTP error (400+)

Example:

```
s_validate_endpoint "example.com" "80"
s_validate_endpoint "api.example.com" "8080" "sozu-prod"
```

1.26 Complete Workflow Example

Here's a complete example of setting up a web service through Sozu:

```
#!/bin/bash
source /path/to/lib_sozu.sh

# 1. Create a cluster
s_cluster_add "myapp-cluster" "random"

# 2. Add backend servers
s_backend_add "myapp-cluster" "server-1" "192.168.1.10:8080"
s_backend_add "myapp-cluster" "server-2" "192.168.1.11:8080"

# 3. Create a listener (optional, if not using default port 80)
s_listener_http_add "0.0.0.0:8080"

# 4. Create frontend binding
s_frontend_http_add "0.0.0.0:80" "myapp.example.com"
 ↪  "myapp-cluster"

# 5. Validate the endpoint
s_validate_endpoint "myapp.example.com" "80"

# 6. List everything to verify
echo "=== Clusters ==="
s_cluster_list

echo "=== Listeners ==="
s_listener_list

echo "=== Frontends ==="
s_frontend_list
```

1.27 Testing

A comprehensive test suite is provided in `test_lib_sozu.sh`:

```
# Make test script executable
chmod +x test_lib_sozu.sh

# Run tests (requires Docker and Sozu container running)
./test_lib_sozu.sh
```

The test script will: 1. Create test cluster, backend, listener, and frontend 2. Verify each operation succeeded 3. Clean up all test resources 4. Report pass/fail statistics

1.28 Output Formatting

All functions return raw Sozu output by default. An example formatting script is provided in `example_sozu_format.sh` showing how to parse and format list outputs.

To use the formatting examples:

```
source example_sozu_format.sh

# Show formatted status overview
show_sozu_status

# Or format individual lists
format_cluster_list
format_listener_list
format_frontend_list
```

1.29 Error Handling

All functions follow consistent error handling:

- **Success:** Print success message with operation details, then raw output
- **Failure:** Print error message describing the failure
- **Return codes:**
  - `0` = success
  - `1` = docker/compose command failed
  - `2` = sozu command failed

Example error output:

```
ERROR: Failed to add cluster 'web-cluster': cluster already exists
```

Example success output:

```
SUCCESS: Cluster 'web-cluster' added with policy 'random'
[raw sozu output]
```

1.30 Integration with HPS System

These functions are designed to integrate with the HPS system:

- Prefix `s_` indicates service-level functions
- Can be called from IPS or TCH/SCH nodes
- Compatible with existing HPS logging and config functions
- Follow HPS function documentation standards

To integrate with HPS config storage:

```
# Store cluster config in cluster_config
cluster_config "set" "sozu_cluster_myapp" "myapp-cluster"
```

```
# Store backend config in host_config
host_config "00:11:22:33:44:55" "set" "sozu_backend"
 ↪  "192.168.1.10:8080"
```

1.31 Notes

- All functions use `docker exec -i` for non-interactive execution
- The `-u root` flag is only used for `s_cluster_add` as required by Sozu
- Config file path `/etc/sozu/config.toml` is hardcoded in the container
- Sudo is not used internally - run the entire script with sudo if needed
- Container verification is performed before each operation
- Default container name is `sozu` but can be overridden per function call
- Functions no longer depend on docker compose, making them more portable

1.32 Load Balancing Policies

Currently tested policies: - `random` - Random backend selection - `roundrobin` - Round-robin backend selection

Note: Other policies may be available but are untested.

1.33 Future Enhancements

Potential additions: - HTTPS listener and frontend functions - TLS certificate management - Backend health check functions - Metrics and statistics retrieval - Container availability checking - Configuration backup/restore - Idempotency checks (don't re-add existing resources)

1.34 License

This library is part of the HPS System project and follows the same open source license.

Repository: https://github.com/OpenDigitalCC/hps-system/ # Storage Provisioning

1.35 Overview

This manual documents the storage provisioning system for debugging purposes. Under normal operation, OpenSVC handles all storage provisioning automatically. This guide is for system administrators who need to manually execute storage operations for troubleshooting or testing.

1.36 Prerequisites

All operations require the node functions to be sourced first:

```
# Source the functions library
. /srv/hps/lib/node_functions.sh
```

1.37 Architecture

1.37.1 Node Types

- **TCH** (Thin Compute Host) - Virtual machine hosts that request storage
- **SCH** (Storage Cluster Host) - Physical storage nodes that provide zvol/iSCSI resources

1.37.2 Components

- **ZFS zvols** - Block devices for storage volumes
- **LIO/iSCSI** - Network block device targets
- **OpenSVC** - Cluster orchestration layer

1.38 Manual Operations

1.38.1 1. Check Available Storage Capacity

Query available space on local storage pool:

```
# Get available bytes
storage_get_available_space

# Example output: 51504332800 (approximately 48GB)
```

Convert human-readable sizes:

```
# Parse capacity string to bytes
storage_parse_capacity "100G"

# Example output: 107374182400
```

1.38.2 2. Provision a Storage Volume

Create a complete storage volume with zvol and iSCSI target:

```
storage_provision_volume \
  --iqn iqn.2025-09.local.hps:vm-disk-001 \
  --capacity 100G \
  --zvol-name vm-disk-001
```

What happens:

1. Validates host type is SCH
2. Retrieves local zpool name
3. Checks available capacity
4. Creates ZFS zvol
5. Creates iSCSI target with backstore
6. Configures LUN mapping
7. Sets up demo mode (no authentication)

Requirements:

- Must run on SCH host
- Sufficient capacity in zpool
- Valid IQN format
- Unique zvol name

1.38.3 3. Remove a Storage Volume

Delete both iSCSI target and underlying zvol:

```
storage_deprovision_volume \
  --iqn iqn.2025-09.local.hps:vm-disk-001 \
  --zvol-name vm-disk-001
```

What happens:

1. Validates host type is SCH
2. Deletes iSCSI target configuration
3. Removes backstore
4. Destroys ZFS zvol

1.39 Low-Level Operations

1.39.1 ZFS Volume Management

Direct zvol operations via the storage manager:

```
# Create zvol
node_storage_manager zvol create \
  --pool ztest-pool \
  --name my-volume \
  --size 50G

# Delete zvol
node_storage_manager zvol delete \
  --pool ztest-pool \
  --name my-volume

# List zvols in pool
node_storage_manager zvol list --pool ztest-pool
```

```
# Check if zvol exists
node_storage_manager zvol check \
  --pool ztest-pool \
  --name my-volume

# Get zvol information
node_storage_manager zvol info \
  --pool ztest-pool \
  --name my-volume
```

1.39.2 iSCSI Target Management

Direct LIO/targetcli operations:

```
# Start target service
node_storage_manager lio start

# Stop target service
node_storage_manager lio stop

# Check service status
node_storage_manager lio status

# Create iSCSI target
node_storage_manager lio create \
  --iqn iqn.2025-09.local.hps:target-name \
  --device /dev/zvol/pool/volume

# Create with ACL (optional)
node_storage_manager lio create \
  --iqn iqn.2025-09.local.hps:target-name \
  --device /dev/zvol/pool/volume \
  --acl iqn.2025-09.initiator:client1

# Delete iSCSI target
node_storage_manager lio delete \
  --iqn iqn.2025-09.local.hps:target-name

# List all targets
node_storage_manager lio list
```

1.40 OpenSVC Integration

1.40.1 Service Structure

The storage-provision service provides two tasks:

```
# Check capacity on all storage nodes
om storage-provision run --rid task#check-capacity --node=\*

# Provision on specific node
om storage-provision instance run \
  --rid task#provision \
  --node=SCH-001 \
  --env IQN=iqn.2025-09.local.hps:vm-disk-001 \
  --env CAPACITY=100G \
  --env VOLNAME=vm-disk-001
```

1.40.2 Viewing Logs

```
# Get session ID from run output
om storage-provision run --rid task#check-capacity --node=\*
# Output shows: OBJECT NODE SID

# View logs for specific session
om storage-provision log --filter SID=<session-id>
```

1.41 Troubleshooting

1.41.1 Common Issues

"ERROR: This host type is 'XXX', not 'SCH' "

- Storage operations only allowed on Storage Cluster Hosts
- Verify: `remote_host_variable TYPE`

"ERROR: Insufficient space"

- Requested capacity exceeds available space
- Check: `storage_get_available_space`
- Reduce capacity or free up space

"Zvol already exists"

- Volume name conflict
- List existing: `node_storage_manager zvol list`
- Choose different name or delete existing

"Failed to create backstore"

- Backstore name already in use
- List: `node_storage_manager lio list`
- Delete conflicting target first

1.41.2 Debugging Steps

1. **Verify host configuration:**

```
remote_host_variable TYPE
remote_host_variable ZPOOL_NAME
```

2. **Check available resources:**

```
storage_get_available_space
zpool list
node_storage_manager zvol list
node_storage_manager lio list
```

3. **Test connectivity:**

```
node_storage_manager lio status
systemctl status target
```

4. **Review logs:**

```
journalctl -u target -n 50
tail -f /var/log/messages
```

1.42 Manual Cleanup

If automated cleanup fails, manually remove resources:

```
# 1. Delete iSCSI target
targetcli /iscsi delete iqn.2025-09.local.hps:target-name

# 2. Delete backstore
targetcli /backstores/block delete backstore-name

# 3. Save configuration
targetcli saveconfig

# 4. Delete zvol
zfs destroy pool-name/volume-name

# 5. Verify cleanup
zfs list -t volume
targetcli ls
```

1.43 Safety Notes

- Always verify host type before provisioning
- Check capacity before creating large volumes
- Ensure unique IQN and volume names
- Use deprovision function for proper cleanup
- Monitor zpool space regularly

1.44 References

- Functions location: `/srv/hps/lib/node_functions.sh`
- OpenSVC service: `storage-provision`
- Target config: `/etc/target/saveconfig.json`
- Logs: `journalctl` and `om storage-provision log` ## Storage configuration

This section describes how storage is provisioned and managed within HPS.
It covers the role of ZFS, iSCSI exports, and the functions used to configure storage cluster hosts (SCHs) and thin compute nodes (TCNs).

1.44.1 Overview

HPS provides storage to diskless compute nodes via **ZFS-backed iSCSI exports**.
Rather than replicating data at the storage back-end, redundancy is achieved through **client-side RAID** on the TCNs. This design keeps the storage back-end simple ("just a bunch of block devices") and allows flexible RAID layouts on the client.

1.44.2 Functions and building blocks

Several library functions are provided to initialise and manage storage nodes:

- **remote_log**
  Sends log output from a remote node back to the provisioning system.
- **remote_cluster_variable** and **remote_host_variable**
  Used to read and update cluster-level and host-level configuration variables across the environment.
- **initialise_opensvc_cluster**
  Prepares an OpenSVC cluster context for managing storage resources.
- **load_opensvc_conf**
  Loads the OpenSVC configuration to ensure cluster services are consistent across nodes.
- **ZFS install and configure functions**
  Automate installation of ZFS, creation of zpools, and export of zvols for use as iSCSI devices.

Additional functions can be added over time. These are distributed to remote nodes (e.g. TCNs and SCHs) and sourced locally so they can operate with the same provisioning logic as the HPS controller.

1.44.3 Host configuration variables

The iSCSI devices used by each thin compute node are defined in the host configuration. Two key variables are:

- **ISCSI_ROOT_0**
- **ISCSI_ROOT_1**

These identify the iSCSI targets that provide the root disks for the TCN.
When a TCN is first configured, these variables are created and stored in its host config.

Future expansions will allow additional variables such as **ISCSI_DATA_N** to define data disks for workloads running on the TCN.

1.44.4 Provisioning workflow

1. **SCH setup**
   Storage cluster hosts are provisioned with ZFS and OpenSVC. ZFS zpools and zvols are created as needed for iSCSI targets.
2. **TCN request**
   When a thin compute node is being installed, the provisioning system determines its storage requirements and allocates ISCSI_ROOT variables.
3. **OpenSVC orchestration**
   OpenSVC is instructed to create the relevant iSCSI targets on one or more SCHs. These zvol-backed targets are exported over the storage network.
4. **Client RAID**
   The TCN combines the iSCSI devices into a RAID set (e.g. RAID1 using `mdadm`) to provide redundancy. This allows failover if one SCH becomes unavailable.
5. **Installation**
   The TCN OS is installed directly onto the iSCSI-backed RAID devices, making it fully diskless and resilient to back-end failures.
6. **Expansion**
   Additional iSCSI targets can later be provisioned for data disks and attached to the TCN as needed.

1.44.5 Future extensions

- Automated provisioning of **ISCSI_DATA** targets for application and workload storage.

- Enhanced ZFS tuning (e.g. `volblocksize`, `compression`, `logbias`) to optimise for specific workloads.

- Integration of monitoring hooks so that OpenSVC and HPS can track zpool health and iSCSI target performance.

- Support for flexible RAID scenarios where high-value services use multiple SCHs while low-cost services may only rely on a single disk.

1.44.6 Acronyms

- **SCH** – Storage cluster node (storage host).

- **TCN** – Thin compute node (diskless compute host).

- **ZFS** – Advanced file system and volume manager used for iSCSI backing.

- **iSCSI** – Protocol that exports block storage devices over a TCP/IP network.

- **OpenSVC** – Cluster manager and orchestrator used to control zvol exports and failover behaviour.

1.45 Troubleshooting

Stub: Common problems, causes, and fixes for HPS operation.

1.45.1 logging

See the log files / syslog

1.45.2 Function test

env QUERY_STRING="cmd=generate_opensvc_conf" bash -x /srv/hps-system/http/cgi-bin/boot_manager.sh

2 IPS Functional reference

Functions managed by IPS

Note that functions prefixed “o_” are relayed to nodes, as well as a few select other functions.

2.0.1 `_apkovl_create_alpine_repos_script`

Contained in `lib/functions.d/alpine-tch-build.sh`

Function signature: dce981c22b6fc6de836484d99c4cf6d24b8f8ea405e3e6fae7708b9dcb92393a

2.0.2 Function overview

The function `_apkovl_create_alpine_repos_script()` is primarily used to generate a shell script that sets up Alpine Linux repository configuration and installs required base packages. The script is created within a temporary directory with permissions that allow for execution.

2.0.3 Technical description

- **name**: `_apkovl_create_alpine_repos_script`
- **description**: This function generates a shell script that configures Alpine repositories and installs the necessary base packages for HPS. The Alpine repository configuration script is created in `local.d` of the provided temporary directory. In case of package installation failure, the shell script outputs an error and exits with a status code of 1.
- **globals**: None
- **arguments**:
  - `$1: tmp_dir` - The provided temporary directory where the script is created.
  - `$2: download_base` - Base URL where the required packages and repositories can be downloaded.
- **outputs**: Outputs a debug message to the HPS log. The function further creates a script in the temporary directory’s `local.d`,
- **returns**: `0` - indicates that the function has completed successfully.
- **example usage**: `_apkovl_create_alpine_repos_script "/tmp/alphine_script" "http://dl-cdn.alpinelinux.org/alpine"`

2.0.4 Quality and security recommendations

1. Implement error-checking on the creation of the temporary directory and handle possible failure scenarios.
2. Sanitize inputs such as the `tmp_dir` and `download_base` to prevent possible code injection vulnerabilities.
3. Improve logging by providing detailed and meaningful logs, especially on failure scenarios.
4. Implementing stricter permissions around the script created in a sensitive directory such as etc.
5. Confirm that `repo_path` returns a proper path before creating the script.
6. Make use of descriptive variable names and maintain consistent code style for better readability and maintainability.

2.0.5 `_apkovl_create_lib`

Contained in `lib/functions.d/alpine-tch-build.sh`

Function signature: 94d89d2a7ab0b2e6f6f5c49f9fc7f5c31b72f5203c0da583552b96ab61660cb2

2.0.6 Function Overview

The function `_apkovl_create_lib` is utilized for creating a bootstrap library for Alpine using a specified temporary directory. The function starts both by creating necessary directories and an executable library file. Additionally, the function adds the newly created library to Linux Bootstrapper's Unit (LBU) protected paths for persistence in an Alpine-specific library path. The function logs debug information in the process and will log an error and return if any step fails.

2.0.7 Technical Description

- **Name**: `_apkovl_create_lib`
- **Description**: This function is responsible for creating a bootstrap library in an Alpine-specific library path using a specified temporary directory (`tmp_dir`). It ensures the necessary directories are created and an executable library file is written. It will also add the library to the LBU protected paths for persistence.
- **Globals**: None
- **Arguments**: [`$1: tmp_dir` - The temporary directory which will be used for creating libraries.]
- **Outputs**: Outputs debug and error logs using the `hps_log` function.
- **Returns**: Returns `1` if any step fails, otherwise returns `0` indicating successful execution.
- **Example Usage**: `_apkovl_create_lib "/tmp/mydir"`

2.0.8 Quality and Security Recommendations

1. Implement additional error checking to verify if the given `tmp_dir` is a valid directory, writable and has enough space before starting the processing.
2. Ensure that user inputs, if any are escaped properly to avoid command injection vulnerabilities.
3. Make sure that all filesystem operations handle symbolic links securely (e.g., avoid race conditions).
4. Consider implementing a rollback mechanism to clean up any partially created directories and files in the event of an error.
5. Optional: Increase verbosity of `hps_log` to include more debugging information (e.g., the actual shell commands being run, their arguments, and their return values).
6. Consider making this function's operations atomic (i.e., they either succeed completely, or the system remains unchanged). This could possibly be done using transactional filesystem operations.

2.0.9 `_apkovl_create_modloop_setup`

Contained in `lib/functions.d/alpine-tch-build.sh`

Function signature: 842181fbbf0920c4f6070433eacf09e160024300c644639ed8d8d84219f4f5de

2.0.10 Function overview

This Bash function, `_apkovl_create_modloop_setup()`, is designed to facilitate the creation of a modloop setup script in the specified temporary directory. This script will be used for downloading and mounting kernel modules. The function begins by defining a series of modules to be loaded. It then creates the modloop setup script, writes instructions for module handling into the script, and finally makes the script executable.

2.0.11 Technical description

- **name:** `_apkovl_create_modloop_setup()`
- **description:** This function creates a modloop setup script inside the specified temporary directory. The script initiates downloads, mounts kernel modules, and handles module loading with dependencies.
- **globals:** [none]
- **arguments:**
  - `$1`: The first argument stands for the temporary directory.
  - `$2`: The second argument represents the base URL to download from.
- **outputs:** The function generates a lot of output, from error logs in case of failure to debugging logs with detailed progress descriptions.

- **returns:** The function will return 1 if it failed to create the directory, make the modloop setup script executable, or to create the local.d directory. It will return 0 if it successfully creates the modloop setup script.
- **example usage:** `_apkovl_create_modloop_setup "/tmp/mydirectory" "http://example.com"`

2.0.12 Quality and security recommendations

1. The function currently lacks input validation, it should perform checks to ensure that the directory paths and URLs passed as arguments are in the correct format and exist.
2. The function could benefit from more detailed logging, for instance, the function could output more specific error messages regarding the stages at which failures occur.
3. It would be beneficial to suppress command outputs that are not crucial to the function's work to make logs easier to read.
4. The function currently does not handle the situation where a module fails to load gracefully, it should be amended to cope with these cases.
5. Functions should be built to only have one exit point for clarity and ease of debugging.

2.0.13 `_apkovl_create_queue_run_script`

Contained in `lib/functions.d/alpine-tch-build.sh`

Function signature: 4a2ed6def4a0d1bd441ad5179c5b5a19d704b202cc10a9cb40a8a1ccc4a170da

2.0.14 Function overview

This bash function, `_apkovl_create_queue_run_script`, is designed to initiate a local script that runs a bootstrap sequence from the HPS (hypervisor) library, if this library can be found. If it cannot, then the function exits with an error message. This sequence runs only after the system is ready, and will output a message when it has successfully completed. The local script that is created, `z-hps-init_run.start`, is stored in the directory specified as the function parameter.

2.0.15 Technical description

- **Name**: `_apkovl_create_queue_run_script`
- **Description**: This function initiates a drive that creates a local script which runs an initialization sequence of HPS after ensuring that the system is ready. The system also handles output and error messages.
- **Globals**:
  - `tmp_dir`: description

- **Arguments**:
  - `$1`: This argument indicates the temporary directory where the localscript will be stored.
- **Outputs**: Messages indicating the start of the HPS init sequence execution, whether the Bootstrap library was found or not, and the end of the sequence execution.
- **Returns**: The function returns the exit status of the last command executed within the function.
- **Example usage**: `_apkovl_create_queue_run_script /tmp/mydir`

2.0.16 Quality and security recommendations

1. A robust function should include more specific error messages to aid in debugging.
2. It would be beneficial to add checks for potential failure points, such as the existence of the temp directory or write permissions on that directory.
3. For security, avoid the use of global variables where possible to reduce the risk of unwanted side effects.
4. Incorporate input validation to check whether the provided parameters are as expected. This can help prevent script malfunction or attempts at malicious script injection.
5. Independent of the context in which it's applied, the permissions set on the file (chmod +x) should be minimally sufficient. Overly permissive settings could be a security vulnerability.

2.0.17 `_apkovl_create_resolv_conf`

Contained in `lib/functions.d/tch-build.sh`

Function signature: 7f68c373f65cdb9fd367f8e973f0778486a609eeeb348c3d3779336610f77261

2.0.18 Function overview

The bash function `_apkovl_create_resolv_conf()` aims to create a new `resolv.conf` file in the specified directory with a passed nameserver. It takes two arguments - a temporary directory and a nameserver, creates a `resolv.conf` file in the given directory and logs the process. If the operation fails, it logs an error message and returns 1.

2.0.19 Technical description

- **Name**: _apkovl_create_resolv_conf
- **Description**: Creates a new `resolv.conf` file with a specified nameserver in a given directory.
- **Globals**: None

- **Arguments**:
  - `$1: tmp_dir`: The directory in which the `resolv.conf` file will be created.
  - `$2: nameserver`: The nameserver that will be written into the `resolv.conf` file.
- **Outputs**: A `resolv.conf` file in the specified directory; log messages of the process or an error message.
- **Returns**: 0 on success, 1 if it fails to create a `resolv.conf` file.
- **Example usage**:

```
_apkovl_create_resolv_conf "/temp/dir" "8.8.8.8"
```

2.0.20 Quality and security recommendations

1. Input validation: Even though not strictly necessary for this function to work, consider validating inputs, like verifying if a directory exists or if a nameserver is valid.
2. Error handling: Instead of just returning 1, consider throwing an exception or giving a more descriptive error output that includes more details of the issue.
3. Code comments: Including comments in the code would help other developers understand the function better, particularly explaining the purpose of the function and its arguments.
4. Logging: It would be beneficial to include more detailed logging, such as logging the successful creation of the `resolv.conf` file.
5. Security improvement: Be cautious with the use of `echo` with redirection > - if not used correctly, it might lead to security vulnerabilities. You should always make sure that user-provided inputs are properly escaped or cleaned up.

2.0.21 `_apkovl_create_runlevel_config`

Contained in `lib/functions.d/tch-build.sh`

Function signature: b3eb801c841e82a67fe14755dc7ba5f92e58b30d32010b4aa81733983d6b0626

2.0.22 Function Overview

The function `_apkovl_create_runlevel_config` is used to create a directory for default runlevels at a given base directory. If the directory creation is successful, it enables local service in the created default runlevel by creating a symbolic link. The function also logs errors in directory creation and symbolic linking, if any, and returns 1 to indicate the errors. If no error is encountered, the function logs successful operation and returns 0.

2.0.23 Technical description

name: `_apkovl_create_runlevel_config`

description: This function helps create a directory for default runlevels in a given base directory. It then enables local service in the created default runlevel. It also logs the outcomes of the directory creation, symlink operation and returns appropriate status code.

globals: None

arguments: - `$1: base_dir` - The base directory path where a new runlevels directory is to be created

outputs: Log messages indicating success or failure of directory creation and symlink operation for local service in default runlevel

returns: The function returns two possible exit codes: - `0` on success - `1` if it fails to create the directory or the symbolic link.

Example usage:

```
_apkovl_create_runlevel_config "/path/to/base_dir"
```

2.0.24 Quality and Security recommendations

1. Consider proper validation and sanitization of the base directory path before using it. This is to prevent directory traversal vulnerabilities where an attacker can specify paths outside the intended base directory.
2. It's crucial to handle errors appropriately. In this case, the function should not only log the error but consider providing a comprehensive report on why the error occurred if possible.
3. Validate whether the directory at `base_dir` and the local service directory `/etc/init.d/local` exist before creating symbolic links to avoid dangling symlinks.
4. Improve logging by including more context or even improving the logs to structured logging with key-value pairs for better traceability and debugging.
5. Ensure to run this function with minimal required privileges to prevent potential security threats. It's a security best practice to always use the principle of least privilege (PoLP).

2.0.25 `_apkovl_create_structure`

Contained in `lib/functions.d/tch-build.sh`

Function signature: de21a4a364feb0c001016f8d48f06a318c8c14aa1115120e71e72b590755ee99

2.0.26 Function Overview

This function, `_apkovl_create_structure`, is responsible for creating a directory structure for Alpine Linux overlay (apkovl). It takes one argument, `tmp_dir`, which specifies the temporary directory where the structure is to be created. The function attempts to create two directories namely, `etc/local.d` and `etc/runlevels/default` within `tmp_dir`. It also sets up a symlink for the local service at boot. Any failure in the creation of directories or symlink results in an error message and halts the execution of the script.

2.0.27 Technical Description

- **Name:** `_apkovl_create_structure`
- **Description:** This function creates a directory structure for apkovl at a specified temporary location. It tries to set up the local service to be enabled at boot.
- **Globals:** None.
- **Arguments:**
  - **$1 (tmp_dir):** The root directory where the apkovl structure is to be created.
- **Outputs:** Logs messages about operation status (debug and error).
- **Returns:**
  - **1:** If failed to create directories or local service symlink.
  - **0:** If execution completes without any error.
- **Example Usage:**

```
temp_directory="/tmp/my_directory"
_apkovl_create_structure $temp_directory
```

2.0.28 Quality and Security Recommendations

1. Check if the argument input (tmp_dir) is not empty. This would prevent potential issues from attempting to create directories at the filesystem root.
2. Check if `tmp_dir` ends with a trailing slash and handle the scenario appropriately to prevent potential directory creation errors.
3. Consider using absolute paths for directory creation to prevent unexpected results due to relative paths.
4. Validate success of each operation, not just directory and symlink creations. This will improve error reporting and make troubleshooting easier in complex setups.
5. Make use of more secure and reliable logging mechanisms for output messages.
6. Employ appropriate file and folder permissions when creating the directories and setting up the symlink to avoid security risks.

2.0.29 `build_dhcp_addresses_file`

Contained in `lib/functions.d/dns-dhcp-functions.sh`

Function signature: e4e89283fe1d64f56e0de04c05dbae29598781f49b67ce46af38b86c9af58337

2.0.30 Function overview

The function `build_dhcp_addresses_file` is a Bash function that builds a list of all the hosts in a network cluster and their associated address details. The function starts by creating a directory if it doesn't exist, and then retrieving a list of all hosts. It processes each host to retrieve and validate their MAC address, IP address, and hostname. The function checks for duplicate MAC and IP addresses, log warnings for duplicates, and stops the process in case of fatal errors like duplicate IP addresses. The details are subsequently written to a temporary file which is then moved to the final location. If successful, a confirmation log is generated with the count of entries created.

2.0.31 Technical description

- **Name:** `build_dhcp_addresses_file`
- **Description:** Builds and writes DHCP addresses of all hosts to a file.
- **Globals:** DHCP_ADDRESSES, DHCP_ADDRESSES_TMP
- **Arguments:** None
- **Outputs:** Logs to standard output at different steps and levels (INFO, WARN, ERROR). Ultimately, outputs a file containing an entry for each host with MAC address, IP address, and hostname, comma-separated.
- **Returns:** `0` if successful, `1` otherwise
- **Example usage:**

```
build_dhcp_addresses_file
```

2.0.32 Quality and security recommendations

1. Make sure to sanitize and validate all inputs and outputs of the function, if it is reused or depended upon in a context with untrusted input.
2. It would be beneficial to add more error handling to verify the success of commands within the function.
3. Adding comments to complex or obscure code sections would improve maintainability.
4. The function could could handle exceptions more gracefully, instead of aborting its operation at the first fatal error.
5. The function should ensure that all sensitive data, such as IP and MAC addresses, is securely handled and not exposed to unauthorized users or services.

2.0.33 `_build_init_sequence`

Contained in `lib/functions.d/node-libraries-init.sh`

Function signature: 0b192da9998d20faaf2d04d9978855d7be31d255e73b831c98a28e20afb3e1cb

2.0.34 Function overview

The `_build_init_sequence` function is designed to initialize a sequence based on provided arguments and specific file contents. It takes six arguments which represent various configurations like base directory, operating system version, type, profile and state. If the rescue argument is provided as true, the function only loads RESCUE inits from the rescue directory. In case of normal mode, it scans and matches metadata of all init files. For every file that matches, it reads non-empty lines excluding comments, trims any leading or trailing whitespace, and adds it to an array. Finally, it outputs an array declaration with selected init files and corresponding actions.

2.0.35 Technical description

- **Name:** `_build_init_sequence`
- **Description:** The function initializes a sequence based on provided arguments and the contents of certain files.
- **Globals:** [`init_files`: a local array storing found .init files, `init_actions`: a local array storing selected actions]
- **Arguments:** [`$1`: the base directory, `$2`: the operating system version, `$3`: the type, `$4`: the profile, `$5`: state, `$6`: rescue mode indicator]
- **Outputs:** An array declaration with the sequence built from init files and corresponding actions
- **Returns:** Nothing.
- **Example usage:**

```
_build_init_sequence "/base/dir" "os_ver" "type" "profile" "state"
 ↪  "rescue"
```

2.0.36 Quality and security recommendations

1. Check that arguments are not empty before using them.
2. Validate inputs before using them in string replacements or file paths.
3. Use secured alternatives for file and directory handling whenever possible.
4. Handle exceptions and error cases more explicitly; provide meaningful error messages if things go wrong.
5. Monitor log messages regularly to detect and analyze possible issues.

2.0.37 `build_yum_repo`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: 55fba2752d840b45670dcc10f8be084d3234f9c87a37a1959fcde6a9247be828

2.0.38 Function Overview

The `build_yum_repo` function checks for changes in RPM packages inside a specified repository. If changes are detected or if there's no previous state, it uses the `createrepo_c` command to create or update the metadata for the YUM repository. Regardless of the outcome, a checksum of current state is stored for future comparison. Outputs and potential errors are reported using `hps_log` helper function.

2.0.39 Technical Description

- **Name:** `build_yum_repo`
- **Description:** The function checks the RPM changes inside the provided repo path. If there are changes or no previous state, `createrepo_c` is used to create or update the repo metadata. Checksums of the current state are saved for future checks.
- **Globals:** None
- **Arguments:** `repo_path` ($1): the path to the repo that needs to be checked and updated.
- **Outputs:** Associated logs with `hps_log` displaying info about events and potential errors.
- **Returns:** 0 if no changes were made or repo created successfully, 1 if the repo path is not provided or doesn't exist, 2 if `createrepo_c` command is not found.
- **Example usage:** `build_yum_repo "${HPS_PACKAGES_DIR}/${DIST_STRING}/Repo"`

2.0.40 Quality and Security Recommendations

1. Consider validating the checksum process and handling any exceptions it might throw. This can improve the function's quality and resilience.
2. Assert the existence of 'createrepo_c' command at the start of the function to fail early if it's not installed, this can save execution time.
3. Look into the security of the checksum generation. Make sure it is robust against potential risks such as spoofing or collision attacks.
4. Validate the structure and content of the repo path argument to prevent potential bugs or security issues related to wrong or malicious inputs.

2.0.41 `cgi_auto_fail`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: 6b67dd57145386562143b5a27ad4c4f0a1e5170fc073f8d4e91738ade5779a4d

2.0.42 Function overview

The `cgi_auto_fail` function is primarily used to detect the client type and based on the detection, it uses different failure methods. Messages are passed as arguments to

the function, and based on the client type, these messages are processed differently. If the client type is `ipxe`, the function `ipxe_cgi_fail` is called; for client types `cli`, `browser`, `script`, unknown, the function `cgi_fail` is called; and for all other scenarios, the function simply logs the error and echoes the message.

2.0.43 Technical description

- **Name**: `cgi_auto_fail`
- **Description**: This function is made to detect the client type and handle failure messages differently based on the client type. The client type is first detected by the function `detect_client_type`, and then care branches deal with the different cases accordingly.
- **Globals**: None
- **Arguments**:
  - `$1: msg` The failure message to be processed.
- **Outputs**: Depending on the client type and the corresponding branch the execution enters, the output could be the execution of the `ipxe_cgi_fail` function, the `cgi_fail` function, or simply echoing the error message and logging it.
- **Returns**: No specific return value.
- **Example usage**: `cgi_auto_fail "Failure message"`

2.0.44 Quality and security recommendations

1. Always sanitize user inputs, especially if these inputs are incorporated into messages or logs.
2. Add comprehensive error handling and logging to help with problem detection and troubleshooting.
3. Regularly update your logging methods to take advantage of whatever logging resources are currently available on the system.
4. Separate the logic of error detection from the error messaging to provide a cleaner and more maintainable code.
5. Consider employing a standard and internationalized method for handling error messages.

2.0.45 `cgi_fail`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: 299e7525f3847f768f357344bca4db2e184ab1855e7b59aadab58dd6c0a5982c

2.0.46 Function Overview

The `cgi_fail` function is part of a Bash script that processes HTTP requests via CGI. It is primarily used to fail an HTTP request and output an error message as

response. This function starts by storing an error message in a variable. Then, it calls the `cgi_header_plain` function to set HTTP response headers to plain text. Afterwards, it logs the error using the `hps_log` function and finally, it prints the error message to the standard output with the echo command.

2.0.47 Technical Description

- **Name:** `cgi_fail`
- **Description:** This function is used for falling an HTTP request and providing an error message as output. It sets HTTP response headers to plain text, logs the error and prints the error message to the standard output.
- **Globals:**
  - VAR: Description depending upon the specific case
- **Arguments:**
  - $1: This argument represents the error message that needs to be displayed. Its description depends upon the particular situation.
- **Outputs:** Outputs the error message to the standard output.
- **Returns:** Does not return a value.
- **Example Usage:**

```
cgi_fail "Error: Unable to process request."
```

2.0.48 Quality and Security Recommendations

1. Always validate and sanitize any input and output data within the function to prevent script injection and other related vulnerabilities.
2. Include comprehensive error handling which provides clear and concise output regarding what went wrong.
3. Avoid disclosing sensitive information in error messages.
4. Abide by the Bash best practices including use of localization, testing and consistent syntax among other things.
5. Regularly update the function to maintain compatibility with updated versions of Bash. Also apply patches and security fixes as necessary.

2.0.49 `cgi_fail`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: 299e7525f3847f768f357344bca4db2e184ab1855e7b59aadab58dd6c0a5982c

2.0.50 Function overview

The `cgi_fail` function is a simple and effective helper function. This function, which is most often found in web server scripts written in Bash, is used to log an error message and send it back as a plain text response. The function first sets a local variable `cfmsg` to the first argument passed in (`$1`). It then calls two other functions: `cgi_header_plain`, which sets the necessary headers for a plain text HTTP response, and `hps_log`, which logs the error message. Finally, it echoes back the error message.

2.0.51 Technical description

2.0.51.1 Name

cgi_fail

2.0.51.2 Description

A Bash function used to log an error message and send it back as a plain HTTP response.

2.0.51.3 Globals:

None

2.0.51.4 Arguments:

- $1: An error message, `cfmsg`.

2.0.51.5 Outputs

A plain HTTP response containing the error message.

2.0.51.6 Returns

None

2.0.51.7 Example Usage

```
cgi_fail "Failed to retrieve data"
```

2.0.52 Quality and security recommendations

1. Since this function directly uses an argument as a part of an HTTP response, it's essential to sanitize the input to avoid potential HTTP injection attacks.

2. Furthermore, the function does not validate the input. This function would fail if a complex string is passed as the message.
3. It's also important to note that the `cgi_fail` function operates on a 'fail-and-continue' basis; it might be worth considering whether a 'fail-and-exit' approach would be more suitable for your specific use case.
4. As a general practice, error logging like `hps_log` should not be done on production systems unless necessary, as it could potentially expose sensitive data.

2.0.53 `cgi_header_plain`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: 47beb816d6fe2217fea69c46b2b01752c6619fa18faae7cd2631960f947f2a66

2.0.54 Function overview

The function, cgi_header_plain(), is a simple bash function used to set the `Content-Type` header on HTTP responses. This function sets the `Content-Type` to `text/plain`, indicating that the HTTP response body will contain plain text.

2.0.55 Technical description

- **Name**: cgi_header_plain
- **Description**: This function is used to set the HTTP `Content-Type` header to `text/plain`, signalling that the HTTP response body is plain text. It does not take any arguments or global variables, and does not have a return value. It simply prints the header and a blank line to `stdout`.
- **Globals**: None
- **Arguments**: None
- **Outputs**: `Content-Type: text/plain`
- **Returns**: None
- **Example usage**: The function is used as follows: `cgi_header_plain`. This will print `Content-Type: text/plain` directly to `stdout`.

2.0.56 Quality and security recommendations

1. **Input Validation**: Although this function does not take any arguments, careful input validation should always be performed on any input data that could potentially be used in a bash function.
2. **Error Handling**: Consider adding error handling to this bash function to handle potential issues that may arise and make debugging easier.
3. **Test Coverage**: Comprehensive tests should be written for this function to confirm that it behaves as expected in all situations.

4. **Use Secure Coding Practices**: To limit the exposure to potentially harmful bugs, consider applying secure coding practices such as not using `eval`, properly quoting variables, and not relying on external commands when bash built-ins will do.

2.0.57 `cgi_log`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: 9f2c2cf7c0d57e85a08611717b5d691eddf235f096bbc311bf9d58541f0c77b3

2.0.58 Function overview

The function `cgi_log()` is designed to output log messages with a timestamp into a file. It takes a string as an argument and appends it to the log file with a timestamp for traceability purposes.

2.0.59 Technical description

- **name:** cgi_log
- **description:** The function accepts a string as an input and writes it to the log file (/var/log/ipxe/cgi.log) with a timestamp. This provides a chronological record of all the logging information.
- **globals:** None
- **arguments:**
  - [$1: msg] Log message as string
- **outputs:** Appends the log message with a timestamp to the /var/log/ipxe/cgi.log.
- **returns:** Not applicable.
- **example usage:** `cgi_log "This is a test message"`

2.0.60 Quality and security recommendations

1. **Input Validation**: Always validate the input (msg) before using it. This will prevent log injection attacks.
2. **Log Rotation**: To manage the size of the logs properly, implement some type of log rotation either via a Bash script or a system utility.
3. **Permissions**: Ensure appropriate permissions are set on the log file to prevent unauthorised access or alteration of the logs.
4. **Sensitive Information**: Be cautious while logging messages as it should not include any sensitive data like passwords which can be exposed through logs.
5. **Error Handling**: Consider adding error handling to log any potential errors when trying to write to the log file.

2.0.61 cgi_param

Contained in lib/functions.d/cgi-functions.sh

Function signature: 85408b6df1acbfa275ec1da76b313025e087b22ded762bda3f063e12f6b557f6

2.0.62 Function overview

The cgi_param function in Bash is used to interpret and manipulate parameters from a QUERY_STRING in a CGI context. The function provides different actions based on the input command provided to it. The available commands are 'get', 'exists', and 'equals'. The function begins by checking if QUERY_STRING has been parsed. On its first run, it decodes the QUERY_STRING into key-value pairs and stores them for subsequent use. It then uses the chosen command to return specific outputs.

2.0.63 Technical description

- **Name:** cgi_param
- **Description:** A function for interpreting and manipulating parameters from a QUERY_STRING in a CGI context.
- **Globals:**
  - __CGI_PARAMS_PARSED: It indicates whether the QUERY_STRING has already been parsed into key-value pairs.
  - CGI_PARAMS: Associative array that holds the decoded key-value pairs from the QUERY_STRING.
- **Arguments:**
  - $1: Command: The action that the function should perform - 'get', 'exists', or 'equals'.
  - $2: Key: The name of the CGI parameter to be retrieved, checked for existence, or compared to a value.
  - $3: Value: An optional value to compare with the value of the specified CGI parameter. Only associated with the 'equals' command.
- **Outputs:** The function outputs either the value of the specified parameter (if the command is 'get'), or specific exit code that indicates the function's success.
- **Returns:** The function generally does not have a numeric return value, exits are primarily via 'return'.
- **Example Usage:**

```
cgi_param get username
cgi_param exists email
cgi_param equals state active
```

2.0.64 Quality and security recommendations

1. Validate all input: Although the function performs some validation on the key inputs, also ensuring the validation of command and value inputs could increase

security.
2. Regularly update the function to meet current best practices and standards.
3. Avoid using global variables: Global variables can be altered anywhere in your script, leading to unpredictable results. Use if only necessary and ensure to control their state.
4. Provide clear error messages: The function could provide more detailed error messages that clearly indicate what the user did wrong.
5. Secure the function against SQL injection: The function currently does no specific check against SQL injection attacks. Consider implementing features to guard against this.
6. Consider edge cases: Always consider possible edge cases in order to ensure that the function behaves reliably.

2.0.65 `cgi_require_param`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: 36f5ec93576d12d13f6f9df4d48da7fb44f5cdb6ffb45d870e5df8a9f1d7eb07

2.0.66 Function Overview

The function `cgi_require_param` is a Bash function which retrieves a specific parameter from the CGI environment. The function accepts one argument: the name of the parameter to fetch. If the specified parameter does not exist, or if it's value is empty, the function will automatically fail and exit the script, displaying an error message. Otherwise, it will print the value of the parameter.

2.0.67 Technical Description

- **Name:** `cgi_require_param`
- **Description:** This function retrieves a named parameter from the CGI environment. If the named parameter does not exist or its value is empty, the function fails and exits the execution of the script with an error message. If successful, it prints the parameter's value.
- **Globals:** None.
- **Arguments:**
  - `$1`: `param_name` - The name of the parameter to be retrieved from the CGI environment.
- **Outputs:** Prints the value of the requested parameter if it exists and is not empty. Otherwise, prints an error message.
- **Returns:** Nothing explicitly. However, if the precondition is not met (the specified parameter exists and is not empty), it terminates the script prematurely with an exit status of 1.

- **Example Usage:** `cgi_require_param "username"` This command will attempt to retrieve the `username` parameter from the CGI environment, fail and exit if it does not exist or is empty, and print the value if it is found and not empty.

2.0.68 Quality and Security Recommendations

1. This function could be improved by handling different kinds of failure with different exit codes. This would allow scripts using this function to differentiate between a missing parameter and an empty parameter.
2. From a security perspective, this function could also be improved by validating the requested parameter's value before printing it. This could prevent potential security risks associated with outputting unvalidated user-supplied input.
3. Logging: It could be helpful to log the actions, particularly the failure cases.
4. Robustness: This function relies on the presence and correctness of other functions (`cgi_param` and `cgi_auto_fail`). Ensuring these dependencies are robust and reliable will improve the overall quality of this function.

2.0.69 `cgi_success`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: 514b6a0abc4d3054c170946dff0ca831cafb51ed3abc448fdd4f1887a82a8de6

2.0.70 Function overview

The bash function `cgi_success()` is used to deliver a success message in the context of a CGI (Common Gateway Interface) script. The script calls another function `cgi_header_plain` first, which is likely to format the message header in a specific way. After the header function, it proceeds to echo (or output) a message, without newline, as defined by the first argument `$1`.

2.0.71 Technical description

- **name**: `cgi_success`
- **description**: A simple bash function that is used within CGI scripts to format and display success messages. It does this by first calling another function `cgi_header_plain` to set the appropriate header, then echoes the input message - without a newline at the end.
- **globals**: None
- **arguments**:
  - `$1`: The success message to display.
- **outputs**: Message as per the provided argument `$1`, outputted without newline at the end.
- **returns**: Nothing

- **example usage**:

```
cgi_success "File uploaded successfully"
```

2.0.72 Quality and security recommendations

1. Always sanitize the input to the function, especially if it is coming from an untrusted source.
2. It's recommended to use printf instead of echo, as echo may have different behavior on different systems.
3. Make sure to handle any potential errors that may arise from the internal `cgi_header_plain` function.
4. Consider adding function return checks where this function is used, to handle any execution failures appropriately.
5. Check for any potential sources of shell expansion or code injection within the function argument.

2.0.73 `cgi_success`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: 514b6a0abc4d3054c170946dff0ca831cafb51ed3abc448fdd4f1887a82a8de6

2.0.74 Function Overview

The function `cgi_success()` is used in Bash scripting within the CGI (Common Gateway Interface) context. It first calls the function `cgi_header_plain` and then prints the message provided as its argument without trailing newlines. This function is commonly used to send HTTP responses with custom success messages as its payload.

2.0.75 Technical Description

- **name:** `cgi_success`
- **description:** This function is part of a CGI script. It calls the function `cgi_header_plain` to output the standard Content-Type header for a plain text file and then uses `echo -n` to print the input argument (sparse) without a trailing newline.
- **globals:** None
- **arguments:**
  - `$1:` The message to display as the success message. This can be any string.
- **outputs:**
  - The function outputs the plain text header and the success message.
- **returns:** Not applicable as the function doesn't explicitly return anything.
- **example usage:**

```
cgi_success "Operation completed successfully."
```

2.0.76 Quality and Security Recommendations

1. Ensure proper sanitization of the input to the `cgi_success` function to prevent any potential security risks, such as code injection.
2. Validate the argument length and type before passing it to `echo` to prevent unexpected behaviour or runtime errors.
3. As with any CGI script, consider potential concurrency issues and use appropriate synchronization mechanisms if necessary.
4. To ensure readability and maintainability, always provide a detailed comment documenting what the function does, its inputs and its outputs.

2.0.77 `check_and_download_latest_rocky`

Contained in `lib/functions.d/iso-functions.sh`

Function signature: 95853858409b34c65eb8d886732c7cec3b9e2f99fb6a3b3d95b1bc9d2780fed7

2.0.78 Function Overview

The function `check_and_download_latest_rocky` checks for the latest version of Rocky Linux ISO in the specified architecture (x86_64) and downloads it if it does not already exist. The function also ensures the correct directory structure exists for storing the downloaded ISO. It includes checks for the latest version and whether the ISO already exists in the local directory. If it does not, it then proceeds to download the ISO using curl and displays appropriate messages. Subsequently, the function extracts the downloaded ISO for use with PXE booting.

2.0.79 Technical Description

- **Name:** `check_and_download_latest_rocky`
- **Description:** This function checks for the latest version of Rocky Linux ISO, downloads it if it doesn't already exist, and extracts it for use with PXE.
- **Globals**:
  - `base_url`: base URL for fetching Rocky Linux ISOs
  - `arch`: The architecture of the ISO (default: x86_64)
  - `iso_pattern`: Describes the type of ISO to download (default: minimal)
- **Arguments**: None
- **Outputs**: Log and echo messages regarding the program's progress and operations. These messages include the latest version number, downloading status, ISO availability, and further steps that are carried out by the function.
- **Returns**: None. The function can exit early with a status of 1 if it cannot detect the version.
- **Example Usage**: `bash check_and_download_latest_rocky`

2.0.80 Quality and Security Recommendations

1. Avoid the user of `local` keyword in the global scope, as its behavior might differ across bash versions.
2. Validate and sanitize inputs to prevent command injection attacks.
3. Include error handling for network operations.
4. Consider using a more secure method of downloading the file, potentially through an encrypted connection.
5. Check that requisite permissions exist for creating directories and downloading files.
6. Implement logging for greater visibility for debugging and auditing.
7. Performance improvements could be considered by avoiding unneeded downloads if the ISO already exists. Check for file integrity after download to prevent using a corrupted file.

2.0.81 `check_available_space`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: 106f18dba6188914b70a8af6d1ddf900c6e5702a9f782fb29e86c3906bc878f6

2.0.82 Function Overview

The `check_available_space` function is used to determine whether there is enough available disk space at a given path. The path to be checked is passed to the function as an argument, along with a specified amount of required space in megabytes. If no amount is specified, the function defaults to checking for at least 500MB of available space.

2.0.83 Technical Description

- **Name**: `check_available_space`
- **Description**: This function verifies if enough disk space is available at a given path. The amount of required disk space can optionally be passed as an argument. If not provided, it defaults to checking for 500MB.
- **Globals**: None
- **Arguments**:
  - `$1`: Path to the directory for disk space check. This is a mandatory argument.
  - `$2`: The required disk space in MB. This is an optional argument, if not provided defaults to 500MB.
- **Outputs**:
  - Outputs available disk space in MB at the given path to `stdout`.
  - Logs error messages to `stderr` using `hps_log` if there's an error.
- **Returns**:

  - Returns 0 if the available space is greater than or equal to the required space OR if the path exists and available space can be determined.
  - Returns 1 if the available space is less than the required space OR if the path does not exist or available space cannot be determined.
- **Example usage**:
  - `check_available_space /path/to/directory 1000`: Checks if the directory at /path/to/directory has at least 1000MB of available space.

2.0.84 Quality and Security Recommendations

1. The function currently relies on the availability and behavior of external commands such as `df`, `awk`, and `sed`. The reliance on these commands might introduce potential vulnerabilities and room for behavior inconsistencies across different systems. It is recommended to reduce this dependence by using built-in Bash features wherever possible.
2. Consider adding more comprehensive error handling to deal with situations like the absence of utilities the function depends on.
3. To enhance readability and maintainability, consider refactoring lengthy operations into separate, smaller functions.
4. The function could benefit from input validation. Currently, there is no validation that the required parameters (path and required space) are of the correct format or within plausible ranges.
5. Ensure the function is thoroughly tested with various inputs, including edge and failure cases.
6. Document any assumptions made in the code and any system dependencies.
7. Consider using a static analysis tool to detect potential security vulnerabilities and to enforce a coding standard.

2.0.85 `check_latest_version`

Contained in `lib/functions.d/iso-functions.sh`

Function signature: 3b874dd0168548a5363b9c63f357dab1016d452e1155213b1190930b50bb44c5

2.0.86 Function overview

The bash function `check_latest_version()` is designed to check the latest version of an operating system provided by a manufacturer for a specified CPU architecture. The function currently supports operating systems hosted on `rockylinux.org`. Inputs include the CPU architecture, the manufacturer, and the operating system name. Outputs will be the latest version of the specified operating system or appropriate error messages.

2.0.87 Technical description

- **Name:** `check_latest_version()`
- **Description:** This function checks the provided URL for the latest version of an operating system. Specifically tailored for `rockylinux.org`, the function parses the fetched webpage to find OS version numbers and returns the latest version found.
- **Globals:** No global variables are used in this function.
- **Arguments:** `$1: CPU architecture (Not currently used in function)`, `$2: Manufacturer (Not currently used in function)`,`$3: Operating System name (Used to form URL and output appropriate version or error messages)`
- **Outputs:** Latest version of the operating system or appropriate error messages.
- **Returns:** `0` if the latest version of the operating system is successfully found, `1` otherwise
- **Example Usage:** `check_latest_version x86_64 Intel rockylinux`

2.0.88 Quality and security recommendations

1. Input Validation: Implement input validation for CPU architecture and manufacturer parameters, which are currently unused.
2. Error Handling: Additional error handling could be implemented if the curl command fails due to network issues.
3. Expand OS Support: The function's functionality could be expanded to support other OS providers beyond just `rockylinux.org`.
4. Secure HTTP Transfers: Consider enforcing HTTPS when fetching the HTML to enhance the security of the function.
5. Scrutinize Regular Expressions: Regular expressions used for matching versions could be reviewed and made more robust to prevent potential errors in output.

2.0.89 `cidr_to_netmask`

Contained in `lib/functions.d/network-functions.sh`

Function signature: f7b2c0eaf8bcc538e387c868ecc0741fec832f46b23a43a4bead09f5431bffb5

2.0.90 Function overview

The bash function `cidr_to_netmask` is designed to convert CIDR notation to subnet mask notation. It validates the input format, processes the prefix, divides the prefix into octets, completes the octet and validates the output format. It uses local variables to store temporary values and performs error checking to ensure correct format.

2.0.91 Technical description

- **Name**: `cidr_to_netmask`
- **Description**: This bash function converts CIDR (Classless Inter-Domain Routing) notation (i.e., `10.0.0.0/8`) to subnet mask notation (i.e., `255.0.0.0`).
- **Globals**: None.
- **Arguments**: `$1` is the CIDR notation to convert to a subnet mask, either as a plain number (prefix length) or a CIDR (IP/CIDR) notation.
- **Outputs**: If successful, it outputs the subnet mask in the console, otherwise it logs a warning message.
- **Returns**: It returns 1 if any error occurs, otherwise it returns `0`.
- **Example usage**:

```
# Example usage with CIDR IP notation
cidr=10.99.1.0/24
netmask=$(cidr_to_netmask $cidr)
echo $netmask # output: 255.255.255.0

# Example usage with just prefix length
prefix=16
netmask=$(cidr_to_netmask $prefix)
echo $netmask # output: 255.255.0.0
```

2.0.92 Quality and security recommendations

1. Ensure the function is used with validated input to prevent potential misuse or incorrect results.
2. Consider rewriting or modularizing function for better readability, as it's doing several different tasks.
3. Check the function returns meaningful error messages for all potential error cases to simplify debugging.
4. To enhance security, always use the function in a controlled manner in scripts, avoid exposing it over the network or making it accessible to untrusted users.
5. Integrate unit tests for the function to ensure its correct operation under different scenarios.

2.0.93 `cli_color`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 5b1da9ceb66e5db9828e84c114d322c5fd60f95ee979773f19784a6dbb747c76

2.0.94 Function overview

The `cli_color` function is a utility that takes a color name as an argument and outputs an appropriate color code. The function checks the color global variables to ensure

they're initialized, and handles inputs in a case-insensitive manner. It supports a variety of color names, including "red", "green", "blue", etc. For unrecognized color names, it simply outputs nothing.

2.0.95 Technical description

- **Name**: `cli_color`
- **Description**: This function takes as input the name of a color and returns the corresponding color code in the terminal. If the input color name is not recognized, it returns nothing.
- **Globals**: [`COLOR_RESET`, `COLOR_RED`, `COLOR_GREEN`, `COLOR_YELLOW`, `COLOR_BLUE`, `COLOR_MAGENTA`, `COLOR_CYAN`, `COLOR_WHITE`, `COLOR_BOLD`, `COLOR_DIM`: definitions of terminal color codes]
- **Arguments**: [`$1`: the name of the color that needs to be translated into a terminal color code]
- **Outputs**: The terminal color code corresponding to the color name input. If the color name is unknown, outputs nothing.
- **Returns**: Always returns `0`, indicating a successful termination of the function.
- **Example usage**:

```
msg_color=$(cli_color "red")
echo -e "${msg_color}This text will appear in
 ↪  red${COLOR_RESET}"
```

2.0.96 Quality and security recommendations

1. The function should include a catch-all case statement to handle any color names that are not specifically listed and return an error message instead of silently failing. This will enhance user-friendliness of the function.
2. Consider validating the input color name against a list of supported colors, and throwing an error if the color name is not recognized.
3. These color variables should be encapsulated in a configuration file for better separation of concerns.
4. Encapsulate initialization of color variables into a function to keep the main body of the code clean.
5. Always prefer local variables over globals where possible for data that do not explicitly need to be shared across multiple function calls, to avoid unintended modification.

2.0.97 `cli_info`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 28a3fbfdceeaf8bd49d31ad9ce80e2034af53e3e598755aaaa9e1bba6bb9cd30

2.0.98 Function Overview

The function `cli_info()` is a Bash function primarily used for formatting and presenting information output to the command line. It accepts two parameters; a header and a message. If neither a header nor a message is provided, the function simply returns. If one or both are provided, the function will use ANSI color codes to format the text and output it to the console with a certain format based on the inputs.

2.0.99 Technical Description

- **Name**: `cli_info()`
- **Description**: This function receives two inputs: a header and a message, and prints them to the console. If both a header and a message are given, it prints the header in blue, followed by the message. If only a header is given, it prints the header in blue. If only a message is given, it prints the message without any color. If neither are given, it simply returns.
- **Globals**: [`COLOR_RESET`: The ANSI color code to reset the color, `COLOR_BLUE`: The ANSI color code for blue]
- **Arguments**: [`$1: The header`, `$2: The message`]
- **Outputs**: Printed header and/or message to the console.
- **Returns**: 0 (successful function execution)
- **Example Usage**: `cli_info "INFO" "This is an informational message"`

2.0.100 Quality and Security Recommendations

1. Check the inputs for potential command injections or illegal characters. Bash does not have native string sanitization functions, but it's generally a best practice to remove or escape any special characters that can have special meanings in unix shells (e.g., `;`, `&`, `|`, etc.).
2. Handle errors by logging them and exiting gracefully, rather than just returning.
3. Incorporate a logging mechanism to record all outputs for future troubleshooting.
4. When printing user-provided inputs if they contain any sensitive data, make sure to properly redact this data to prevent information leaking.

2.0.101 `cli_init_colors`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: e0b1c767cca4df9b582c6ff8ad624787bdca5f882be1cc1fd7aa5eb774969d6b

2.0.102 1. Function Overview

The Bash function `cli_init_colors()` is used to initialize the color settings for terminal text output. The function checks if colors should be used, and if so, assigns ANSI color codes to various variables. If colors should not be used, these variables are set to empty strings. The settings are then exported for use in subshells. No arguments are needed for this function.

2.0.103 2. Technical Description

- **Name:** `cli_init_colors()`
- **Description:** This function initializes ANSI color codes for terminal output or assigns empty strings to color variables if no color should be used. The variables containing color settings are then made available to subshells.
- **Globals:**
  - NO_COLOR: Determines if color should be used. If this variable is set, no color will be used.
- **Arguments:** None
- **Outputs:** The function sets and exports the color settings variables.
- **Returns:** Always returns `0`, which in Bash script implies successful execution.
- **Example usage:** “‘ #! /bin/bash

  cli_init_colors echo “$COLOR_REDTh istextis red${COLOR_RESET}” “‘

2.0.104 3. Quality and Security Recommendations

1. Always quote variable assignments to avoid word splitting and pathname expansion. Especially in circumstances where either could lead to a security vulnerability.
2. Check the existence and value of `${NO_COLOR:-}` more robustly. If it’s unintentionally set as a non-empty string that doesn’t explicitly disable color, the function will prevent the use of colors.
3. For safety, consider checking if the terminal supports specific colors before using them. Not all environments support the full ANSI color range.
4. Validate any user input that may affect the color rendering or be put into the NO_COLOR variable. Inadequate validation may lead to ‘code injection’ where attackers could insert malicious code.
5. The function always returns `0`, indicating success. However, it could be useful to implement error codes to signal when an issue arose during the function’s invocation, such as color-code setting failure.

2.0.105 `cli_note`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 1b1a001917f29b7c34dd816c043b9b720e5299789d848d1a2d0f19f67d867d1f

2.0.106 Function overview

The `cli_note` function is a simplistic Bash function intended to provide an easy way to make noted messages more visual in output. The function takes in one argument, namely a message, which it then formats in a specific way before printing to the user.

2.0.107 Technical description

```
- **Name**: cli_note
- **Description**: This function takes a message as an argument and logs it, formatted
- **Globals**:
    - None
- **Arguments**:
    - `$1`: message
- **Outputs**: Prints "Note: message" to stdout.
- **Returns**: Always returns 0 to signify successful execution of the function.
- **Example Usage**:
```bash
cli_note "This is a test note"
```
```

The above code will print to the screen:
`Note: This is a test note`

2.0.108 Quality and security recommendations

1. **Input Validation**: Always validate inputs. In the case of `cli_note`, ensure the input is a string before processing.
2. **Error Handling**: In the event of an error, ensure that the function fails gracefully. A suitable return code and an error message could be beneficial in any situation where the message is not a string or is not provided.
3. **Internationalisation**: If the application operates in multiple languages, make sure the “Note:” prefix can be translated for international users.
4. **Documentation**: Make sure to document the function’s behavior and any edge cases for others who may use or maintain your code.
5. **Testing**: Continuously test the function with various inputs to ensure it behaves as expected, and consider edge cases and potential misuse of the function. For instance, providing special characters as input for this function could potentially cause issues.

2.0.109 `cli_prompt`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: f49e68ce40ea9bb18b1ae2b2549fbaf0b8bc3bef5f929a8a9764769dc4ce04d5

2.0.110 Function overview

The `cli_prompt` function is a Bash utility function that prompts the user for input on the command line, sets a default value if the user inputs nothing, and validates the user's input against a specified regular expression if one is provided. If the user's input does not match the regex, it logs an error and returns 1. If the user's input is valid or no validation is necessary, it echoes the input for use in the script or environment, then returns 0.

2.0.111 Technical description

```
cli_prompt() {
  ...
}
```

name: `cli_prompt`

description: This function prompts the user for input on the command line with the ability to set a default value and validate the input against a given pattern.

globals: None

arguments: `$1`: `prompt` - The string to display to the user on the command line. `$2`: `default` - The default value to apply if the user's input is empty. `$3`: `validation` - The pattern to validate the user's input against. `$4`: `error_msg` - The error message to log if the user's input is invalid.

outputs: If the user's input is valid, it outputs the input.

returns: If the user's input is invalid, it returns 1. If the user's input is valid, it returns 0.

example usage:

```
cli_prompt "Enter your name" "John Doe" "^[a-zA-Z ]*$" "Name can
 ↪  only contain letters and spaces"
```

2.0.112 Quality and security recommendations

1. Always provide meaningful prompts to guide the user on what to input.
2. Use the `default` parameter judiciously. If the choice has substantial effect or if the user input is sensitive, a default might not be a good idea, you want to ensure the user consciously inputs information.
3. Regulate the use of regular expressions in the `validation` variable. This could be a point of vulnerability if misused. Be careful to filter out any sensitive or malicious input.

4. Always provide a meaningful `error_msg` which would guide the user on the right input to provide when they make an error.
5. Be careful when echoing out the user input, as this could lead to some potential command injection vulnerabilities. Always sanitize your inputs and do not trust user input blindly.

2.0.113 `cli_prompt_yesno`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: f47ec2eb44075d167f6f356ea8bd4fc03757eb9a99d783f80e432210369044ff

2.0.114 Function Overview

The bash function `cli_prompt_yesno()` is a utility that has been developed to interact with the user through a command line interface (CLI). It prompts the user to provide a 'yes' or 'no' response, with the possibility to set a default response. It is handy for confirming actions or choices in automation scripts. The function takes in a prompt message and a default value ('y' for yes, 'n' for no) as inputs, and outputs the normalized user response.

2.0.115 Technical Description

- **Name:** `cli_prompt_yesno`
- **Description:** A Bash function that outputs a prompt to a user in a command line interface and waits for a 'yes' or 'no' response. It has the capability to use a default response if the user just hits the enter key without providing any input. The user's input is normalized to 'y' or 'n', and any other input results in an error message and an unsuccessful function exit.
- **Globals:** None
- **Arguments:**
  - `$1: prompt` This represents the question or prompt message to be displayed to the user.
  - `$2: default` This is the default value (either 'y' for yes, or 'n' for no) to be used if the user just hits the enter key without providing any input.
- **Outputs:** The function will normalize the user input to either 'y' or 'n' or an error message, which it echoes to stdout.
- **Returns:**
  - `0` if the user input is valid (i.e., either 'y' or 'n' even after applying the default value).
  - `1` if the user input is invalid (neither 'y' nor 'n').
- **Example usage:** `bash if cli_prompt_yesno "Are you sure you want to proceed?" "n"; then echo "Proceeding, ..."`

`else echo "Abort operation!" fi` ### Quality and Security Recommendations

1. Add input validation for the `default` argument, to ensure it's either 'y' or 'n'. Any other inputs should lead to a function failure.
2. Internationalize the function by allowing translation of the 'yes' and 'no' strings.
3. Make sure the prompt string is safely escaped to prevent command injection attacks.
4. Consider returning distinct error codes for different types of errors, such as input validation errors and invalid user inputs, to allow the calling code to respond appropriately.

2.0.116 `cli_select`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: c980cde4828d1b931c6014c0b115b19f394558bf05df9574129aae3aff155311

2.0.117 Function overview

The `cli_select` function is a command-line interface tool in Bash. It provides an interactive menu to the user, allowing for the selection of a choice from an array of provided options. When a valid choice is entered, it is printed and the function returns. In case of an invalid selection, an error message is logged and the selection process repeats. If a SIGINT signal is detected (e.g., user inputs Ctrl+C), the function terminates and returns 1.

2.0.118 Technical description

- **Function Name**: `cli_select`
- **Description**: This function generates an interactive menu that allows a user to select an option from an array of given choices. The function will print the selected choice or an error message in case of an invalid selection. If the user sends a SIGINT signal, the function will terminate immediately.
- **Globals**: None
- **Arguments**:
  - `$1`: This is the prompt message displayed to the user.
  - `$2-N`: These arguments represent the available options for the user to select from.
- **Outputs**:
  - Valid selection: prints the chosen option to stdout.
  - Invalid selection: calls the `hps_log` function, which presumably logs the error message "Invalid selection".
- **Returns**:
  - 0: when a valid selection has been made and printed.

  - 1: when the user breaks the selection with a SIGINT signal.
- **Example usage**:

```
cli_select "Choose a fruit" "Apple" "Banana" "Cherry"
```

2.0.119 Quality and security recommendations

1. Consider adding input validation for the prompt and options arguments. This will ensure they do not contain malicious or unexpected characters.
2. Currently the function relies on the `hps_log` function to log errors. Make sure that this function properly sanitizes the log message to prevent log injection attacks.
3. There could be an infinite loop if the `hps_log` function doesn't stop the script. Add a maximum number of attempts to prevent this.
4. The function echo's the user's choice. Ensure that data is sanitized or properly escaped if it is going to be used in any sort of command to prevent command injection attacks.
5. Look into securely handling signal interrupts besides just SIGINT to ensure a consistent user experience across various scenarios.

2.0.120 `cli_set_active_cluster`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 57af9df89602d50cd32305c1772eda685805080ffdb84fb6a47bc26ab7b1630d

2.0.121 Function Overview

`cli_set_active_cluster` is a Bash function that facilitates the management of clusters within a script or program. This function takes a `cluster_name` as an argument and attempts to set it as the active cluster. If the provided `cluster_name` is the same as the currently active cluster, an information message is printed and the function exits normally. If the user confirms the action (through a yes/no prompt), the function sets `cluster_name` as the active cluster, exports dynamic paths, and commits changes. If any step fails, a corresponding error message is logged and the function returns an error code.

2.0.122 Technical Description

- **Name:** `cli_set_active_cluster`
- **Description:** This Bash function sets a given cluster as active. It logs errors if the `cluster_name` is not provided or if setting the cluster as active or committing changes fail. It informs the user if the `cluster_name` is already the active cluster.
- **Globals:**
  - `VAR: desc` (Not clearly defined from the provided function)

- **Arguments:**
  - `$1: cluster_name` - The name of the cluster to be set as active.
- **Outputs:**
  - Diagnostic and informational messages about the process are printed to STDOUT/STDERR.
- **Returns:**
  - `0` if `cluster_name` is already active or if setting `cluster_name` as active and committing changes succeed.
  - `1` if `cluster_name` is not provided or if setting the cluster as active or committing changes fail.
  - `2` if the user declines setting `cluster_name` as active.
- **Example usage:**

```
cli_set_active_cluster "MyCluster"
```

2.0.123 Quality and Security Recommendations

1. Improve error handling by introducing more granularity in error codes for better debugging.
2. Provide a clearer definition and usage of the global variables if the function depends on them.
3. Consider removing the direct dependency on user input for better automation and script-ability. Options and choices should be passed as parameters instead.
4. Ensure that the cluster names and any strings used in comparison are sanitized to prevent potential command injection attacks.
5. Use more explicit variable names for better code readability and maintainability.

2.0.124 `cluster_config`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 284dd5b86a774afac3a27810601d18c7382904ea1acfc9132f3aa11eee094d53

2.0.125 Function Overview

The `cluster_config()` is a function written in Bash that is used to modify or retrieve the configuration of an existing cluster or a specified cluster that is active or inactive. It takes several parameters: operation type, key, value and cluster. The operation type determines whether to 'get', 'set', or 'exists'. These operations fetch, modify, or confirm the existence of a parameter in the cluster config file respectively. If the cluster parameter is not provided, the function defaults to the active cluster.

2.0.126 Technical Description

- **name:** `cluster_config()`
- **description:** This function performs operations like retrieving, setting, or checking the existence of a configuration key in a specified or default (active) cluster config file.
- **globals:** [VAR: desc]
- **arguments:**
  - `$1`: Operation type, which can be 'get', 'set' or 'exists'.
  - `$2`: Key associated with the desired configuration value.
  - `$3`: Value to be set in case of set operation (optional).
  - `$4`: Cluster to perform operations on (optional, defaults to active cluster).
- **outputs:** Result of the configuration operation; could be the fetched value or the newly set value depending on the operation.
- **returns:** Returns integer signalling success of function - '1' for failure and '2' for an unknown operation.
- **example usage:** To set a value '3' for key 'replica' in the active cluster, usage will be `cluster_config set replica 3`

2.0.127 Quality and Security Recommendations

1. Validate the input provided to the function to avoid any form of command injection.
2. Guard against any possibility of race condition when reading and writing to the cluster files.
3. Use `set -o noglob` to avoid UNIX pathname expansion or globbing which could lead to unexpected results or security issues.
4. Proper error handling should be implemented for the function to allow for debugging and auditing.
5. Test the function across different versions of Bash shell and platforms for compatibility.
6. Implement logging mechanism to track the changes made by each operation.
7. Encryption should be considered when storing sensitive data in the cluster config file.

2.0.128 `cluster_has_installed_sch`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 409c4d1b1b370935d8896096a0eba8d1aee600a78f18df3551d875126c299da2

2.0.129 Function Overview

The `cluster_has_installed_sch` function is a tool that is used to determine if there is at least one School (SCH) host in the cluster that has been installed. It entirely depends on two different function helpers; `host_config` and `list_cluster_hosts`.

It iterates over each host within a cluster, and for each host, it first checks if it is of type SCH, and then if that SCH host is in an installed state. If it finds a SCH host that is installed, it returns 0 to indicate success, else it returns 1.

2.0.130 Technical Description

Name: cluster_has_installed_sch

Description: This Bash function checks if there is at least one installed host of type SCH in a cluster.

Globals:

- None

Arguments:

- No arguments are required for this function.

Outputs:

- It doesn't print anything to stdout.

Returns:

- 0: If an Installed SCH host is found in the cluster.
- 1: If no Installed SCH hosts are found in the cluster.

Example usage:

```
if cluster_has_installed_sch; then
  echo "There is at least one installed SCH in the cluster."
else
  echo "No installed SCH found in the cluster."
fi
```

2.0.131 Quality and Security Recommendations

1. Add input validation measures to ensure that helper functions like `host_config` are receiving valid inputs for a better and safer execution.
2. Consider logging the operation and outcomes. It would help while debugging, in case something goes wrong.
3. You might also want to improve error handling to better account for any common issues that might occur and provide more verbose information.
4. Check if helper functions `host_config` and `list_cluster_hosts` exists before their usage, as they seem to be crucial for this function.
5. To avoid calling multiple times `host_config` function for each host, fetch all config at once then filter on it.

2.0.132 `cluster_storage_init_network`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: fe8eaf471ccdd29f34f50d0dbe88e6d5a67d4d554fa3332f5e695e82244e617f

2.0.133 Function overview

The function `cluster_storage_init_network` is designed to gather information regarding the configuration of storage networks within a computing cluster. The function accepts and validates from the user a variety of parameters - such as the number of storage networks to configure, the base VLAN ID for the storage networks, the storage subnet base, and whether or not to enable jumbo frames - using the `cli_prompt` function. Once all these values are obtained, the function then proceeds to store these values and configure each storage network based on the provided settings. The function also handles the validation of user input - prompting the user once again for incorrect entries - and provides feedback via system log messages.

2.0.134 Technical description

- **Name**: `cluster_storage_init_network`
- **Description**: This function is designed to manage the initialization and setting up of storage networks within a computing cluster.
- **Globals**
  - **Variables**: `cluster_domain`, `num_storage_networks`, `storage_base_vlan`, `storage_subnet_base`, `storage_subnet_cidr`, `enable_jumbo_frames` - These variables are used to hold the system configuration settings as fetched or calculated by the function.
- **Arguments**: None. The function does not accept arguments. All values are gathered via user input.
- **Outputs**: log messages indicating success or failure of initialization.
- **Returns**: `0` on successful completion, `1` on any error.
- **Example usage**:

```
cluster_storage_init_network
```

2.0.135 Quality and security recommendations

1. As the function involves user input, it is important to validate all the data entered by the user. In this case, the validation is handled by `cli_prompt` function, so it should ensure the provided values follow the specified rules.
2. Error-handling mechanisms are already present in the function, but they could potentially be expanded to cover more error cases and give more detailed error messages.
3. Avoid the use of global variables for better code modularity, scalability and testing.

4. Consider breaking the function down into smaller functions that do one specific job, making the code easier to understand, test and maintain.
5. Always keep the software and all of its dependencies up to date to protect against known security vulnerabilities.

2.0.136 _collect_cluster_dirs

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 21a54dd1fed816cbbecd01967b98b68192e6453200d8c224a543b3e6b074b633

2.0.137 Function Overview

The function `_collect_cluster_dirs()` is used to collect all directories from a base directory (excluding symbolic links). The base directory is specified by the global variable HPS_CLUSTER_CONFIG_BASE_DIR. The names of directories are stored in an array, which is passed to the function via a reference variable. If the base directory doesn't exist, the function prints an error message and exits with status zero.

2.0.138 Technical Description

- **Name:** `_collect_cluster_dirs()`
- **Description:** This bash function collects all the directory entries from the base directory specified by the HPS_CLUSTER_CONFIG_BASE_DIR global variable and stores them in the referenced array. It skips symbolic links and any non-directory entries.
- **Globals:**
  - HPS_CLUSTER_CONFIG_BASE_DIR: Description not provided in the sample function. Presumably, this global variable specifies the base directory in which the function searches for directories.
- **Arguments:**
  1. $1: This is a reference to an array. The names of directories found will be added to this array.
- **Outputs:**
  - If the base directory doesn't exist, an error message is written to the standard error output.
  - The function modifies the array passed to it via the reference variable, adding names of directories found.
- **Returns:**
  - The function always returns zero.
- **Example Usage:**

```
# Assume that the global variable HPS_CLUSTER_CONFIG_BASE_DIR is
# already set to some valid directory path
declare -A my_array
```

```
_collect_cluster_dirs my_array
```

2.0.139 Quality and Security Recommendations

1. Document the purpose and usage of global variables used by the function.
2. Consider having the function return non-zero status when an error is encountered, such as when the base directory doesn't exist.
3. Avoid polluting the global scope by unsetting local variables at the end of the function.
4. It would be a good security practice to sanitize inputs to the function or ensure they are of the expected type.
5. Include error handling for cases where the argument passed is not a valid reference to an array.

2.0.140 `commit_changes`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 511b6937ea1ef6712b99a5da762680ed0c87362cbcc9d802fcf9ddecb476ae51

2.0.141 Function Overview

`commit_changes()` is a Bash function designed to process and commit configuration changes for a specified cluster. If a cluster cannot be found or there are no pending configuration changes, the function will log error or informative messages accordingly and exit. If pending configuration exists, each is processed. If any configuration item fails to be set, an error is logged and the function returns 1, indicating an error. After successful processing of all configurations, the pending configuration array is cleared, the DNS and DHCP file updates are triggered, a successful configuration change message is written to the log, and the function returns 0, indicating success.

2.0.142 Technical Description

- **Name**: `commit_changes`
- **Description**: A function to process and commit configuration changes for a specified cluster. Logs messages to indicate the function's status and result.
- **Globals**:
  - `CLUSTER_NAME`: The name of the cluster where the changes will be committed. If not available, an error is thrown and the function returns 1.
  - `CLUSTER_CONFIG_PENDING`: An array storing the configurations to be committed. If empty, a message is logged and the function returns 0.
- **Arguments**: None
- **Outputs**: Logs various messages to indicate the status and result of the commit operation.

- **Returns**:
  - Returns 1 if the cluster name is not available or if setting the new configuration fails.
  - Returns 0 if there are no configuration changes to commit or if the configurations are successfully committed.
- **Example Usage**:

```
commit_changes
```

2.0.143 Quality and Security Recommendations

1. Validate the format and integrity of configuration inputs before processing. This can prevent unexpected behaviour or security issues.
2. Consider adding a dry-run mode where the changes that would be made can be previewed before they are committed.
3. Since this function heavily relies on global variables, consider making it more self-contained by taking both the cluster name and configuration pending as arguments.
4. Enforce proper error handling and logging for each function call and possible failure scenarios.
5. Conduct regular code audits to maintain the function's security and performance.

2.0.144 `config_get_value`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 3770ff8ebc08ace029f2496144d50418a387d3129f3a483c96ff3865b2d9962d

2.0.145 Function overview

The `config_get_value` function is primarily used to fetch configuration values based on a provided key from either a pending or existing cluster configuration. If no configuration value is found, the function returns a default value.

2.0.146 Technical description

Function: `config_get_value`

- **Name:** config_get_value
- **Description:** This function fetches the value of a specified key from the pending or existing configuration of a given cluster. If the key is not found in either configuration, it returns a default value.
- **Globals:** CLUSTER_CONFIG_PENDING - stores temporary configuration items that are yet to be permanently stored; CLUSTER_NAME - a string value representing the name of the cluster.

- **Arguments:**
  1. `$1 (key)`: The key for which to fetch the value.
  2. `$2 (default)`: A default value that is returned if the key is not found in either the pending or the existing configuration.
  3. `$3 (cluster)`: The cluster to check the configuration from. If not provided, the function uses the value of the global variable `$CLUSTER_NAME`.
- **Outputs:** The value of the specified key from either the pending or the existing configuration, or the default value.
- **Returns:** 0 - if the function executes successfully.
- **Example Usage:** `config_get_value "key_name" "default_value"`

2.0.147 Quality and Security Recommendations

1. Confirm that key values are sufficiently unique to avoid unintentional overlapping of configuration variables.
2. Sanitize input parameters to the function (e.g., $1, $2, $3) to prevent potential exploitation of uncontrolled format string vulnerabilities.
3. Use the ${parameter:-word} form for setting default values to prevent unassigned variables from causing errors.
4. Implement improved error handling for scenarios where the configurations cannot be accessed or the specified key cannot be found.
5. Ensure that the global configuration variables, especially `CLUSTER_CONFIG_PENDING` and `CLUSTER_NAME`, are well protected and only modifiable through controlled means.

2.0.148 `configure_kickstart`

Contained in `lib/functions.d/configure_kickstart.sh`

Function signature: b9c974579a4cf0918b1f523ab5546c0fabf4f4fe0d6a0a4ab77a23765ef1cbca

2.0.149 Function Overview

The function `configure_kickstart()` is used to generate a Kickstart configuration file for a specified cluster. Kickstart files are used by the CentOS/Fedora/RHEL boot process to configure new installations. It's a shell script that automates the post-installation process of setting up a customized system.

2.0.150 Technical Description

- **Name**: `configure_kickstart`
- **Description**: This function accepts a cluster name as an argument. It takes this name and creates a Kickstart file customized for that specific cluster. If no cluster

name is provided, it will return an error message and exit. After successfully creating the Kickstart file, it prints the location of the newly generated file.

- **Globals**: [`CLUSTER_NAME` : The name of the cluster, `KICKSTART_PATH` : The location where the Kickstart file will be created]
- **Arguments**: [`$1` : The name of the cluster]
- **Outputs**: Outputs logs descriptive of the processing, including an error message if the cluster name is not provided, a status log regarding kickstart file generation, and the location of the generated Kickstart file.
- **Returns**: Does not return a value and will stop execution if cluster name is not provided.
- **Example Usage**: `configure_kickstart cluster1`

2.0.151 Quality and Security Recommendations

1. Make sure to validate the input parameter to avoid any unwanted results.
2. Always check for existing files before generating a new file to prevent data loss.
3. Ensure the generated files have the correct permissions and owner to prevent potential security breaches.
4. Consider encrypting sensitive data, such as passwords, to enhance security.
5. Develop comprehensive error handling routines to catch and handle any issues during execution.
6. Avoid using hardcoded values (such as “/srv/hps-config/kickstarts”) as much as possible.

2.0.152 `configure_kickstart`

Contained in `lib/functions.d/configure_kickstart.sh`

Function signature: b9c974579a4cf0918b1f523ab5546c0fabf4f4fe0d6a0a4ab77a23765ef1cbca

2.0.153 Function Overview

The Bash function, `configure_kickstart()`, is used to automate the generation of a Kickstart configuration file for a new Linux cluster. This function requires a cluster name as the argument, which is then used to name and generate a Kickstart file in the designated path. This function includes a configuration set up that ensures system requirements, including network configuration, root password, partitioning, and package selection, among others, are met. An error message is displayed when a cluster name is not provided as the argument.

2.0.154 Technical Description

- **Name:** `configure_kickstart()`

- **Description:** This function automates the process of generating a Kickstart configuration file for a Linux cluster using a provided cluster name.
- **Globals:** [`CLUSTER_NAME` : Stores the name of the cluster, `KICKSTART_PATH` : Path where the kickstart file will be generated]
- **Arguments:** [`$1`: Cluster name]
- **Outputs:** Prints status messages during the function execution.
- **Returns:** Returns an error message if the required argument (cluster name) is not provided.
- **Example Usage:** `configure_kickstart test_cluster`

2.0.155 Quality and Security Recommendations

1. Make sure the Kickstart file path is a secure location and has the right permission settings to avoid unauthorized access or alterations.
2. Ensure proper validation is performed on the input cluster name to avoid possible command injection vulnerabilities.
3. The root and user passwords are currently set with placeholder values. Change these values to secure ones before using on production systems.
4. Ensure you handle the potential issue where the cluster name given might already exist, overwriting an existing kickstart file.
5. Consider including a verification step to confirm the successful generation of the Kickstart file or to catch any possible errors during the file creation process.
6. It's recommended to ensure the installation log (`/root/ks-post.log`) is properly secured or even disabled in a production environment to protect system information.

2.0.156 `count_clusters`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 35e810345753544ce73618109f16ad97d33b5185c8ff2d374a8837aa569e5960

2.0.157 Function overview

The `count_clusters` function is a shell command used primarily to gather a count of directories that are considered "clusters". More specifically, it stores these cluster directories into an array and returns the count. If the array is empty, it outputs an error message, and return an exit code of 0 along with a count of 0 to illustrate that no clusters were found.

2.0.158 Technical description

Name: `count_clusters`
Description: This function is used to obtain a count of "cluster" directories.

Globals: [HPS_CLUSTER_CONFIG_BASE_DIR: The base directory where clusters are located]
Arguments: [None]
Outputs: If it cannot find any clusters, it outputs an error message stating “No clusters found in ‘HPS_CLUSTER_CONFIG_BASE_DIR’ ”. If it finds clusters, it outputs the count of clusters.
Returns: It returns 0 regardless of whether it finds any clusters. If it finds clusters, it still returns the number of clusters.
Example Usage:

```
# Count the directories in the base directory.
count_clusters
```

2.0.159 Quality and security recommendations

1. Based on the provided function, there’s an implication that some global variables are used across multiple functions. This could lead to obscure bugs if one function modifies the global variable in a way that another function doesn’t expect. In a safer, clearer, and easier-to-maintain option, those global variables should be turned into arguments to isolate side effects.
2. There is a lack of argument validation in the function. For robust and error-free functioning, it is advisable to check if the calculated directories exist and are indeed directories before counting them.
3. The use of `echo` to communicate errors is not following best practices. Instead, we should consider switching to an logger or exposing error messages through the use of special return codes. This would give users more context on how to resolve the error than just presenting it as a string.

2.0.160 `create_bootstrap_core_lib`

Contained in `lib/functions.d/node-bootstrap-functions.sh`

Function signature: 185ab2e8ed8b7b25b303b8cc7a660c5fe4fc99cf828a17426f8d22cd6d330dba

2.0.161 Function overview

The `create_bootstrap_core_lib` function is used to relay IPS core functions. The function specifically prints out a heading “HPS Bootstrap library”, then it relays a number of HPS functions such as `hps_check_bash_syntax`, `hps_debug_function_load`, `hps_safe_eval`, and `hps_source_with_debug`. After the relay is completed, it echoes “Relay complete” and then begins to write the core functions for HPS node bootstrap and initialization including a URL encoding function.

2.0.162 Technical description

- **Name:** `create_bootstrap_core_lib`
- **Description:** This function is used to relay IPS core functions, specifically starts by printing out a heading “HPS Bootstrap Library”, then relaying a couple of HPS functions and ends by echoing “Relay complete” and beginning to write the core functions for HPS node bootstrap and initialization which includes a URL encoding function.
- **Globals:** No global variables used.
- **Arguments:** The function does not take any arguments.
- **Outputs:** The function spits out some text to stdout including the relaying of HPS functions, ‘Relay complete’, and starting to write the core functions for HPS node bootstrap and initialization.
- **Returns:** The function doesn’t return any value.
- **Example usage:** `create_bootstrap_core_lib`

2.0.163 Quality and security recommendations

1. Incorporate error handling to identify and resolve any potential issues that may occur when declaring and relaying functions.
2. Write unit tests for the function to ensure it’s working as expected at all times.
3. Document the purpose and usage of each relayed function. This will improve the maintainability of the code.
4. Avoid using hard-coded strings. Try to make the function more flexible and not tied to the specific HPS functions.
5. Ensure all the relayed functions are also following security best practices.

2.0.164 `create_config_dnsmasq`

Contained in `lib/functions.d/create-config-dnsmasq.sh`

Function signature: e8e5c3ae68c6edbb046d7d24d90b6f00e97a7553b5f4271d29d93e11655d696a

2.0.165 Function Overview

The `create_config_dnsmasq` function provides the necessary configuration for a dnsmasq server on a specific DHCP IP, ensuring that the specific files necessary for the server exist. By echoing a base configuration for PXE/TFTP/DHCP into a `dnsmasq.conf` file and touching the `dhcp_addresses` and `dns_hosts` files for existence, the function prepares an environment that supports network address translation and DNS/DHCP services.

2.0.166 Technical Description

- Name: `create_config_dnsmasq`

- Description: This function creates the necessary dnsmasq.conf file for a dnsmasq server with a specified DHCP IP, and ensures the existence of the necessary files for DNS/DHCP services.
- Globals: DHCP_IP: The IP address of the DHCP to be configured with dnsmasq.
- Arguments: None
- Outputs: Either en exception error stating that there's no DHCP IP, or a success message that dnsmasq config file has been generated at a specified location.
- Returns: Nil.
- Example usage: `create_config_dnsmasq`

2.0.167 Quality and Security Recommendations

1. Since `${DHCP_IP}` is a global variable, ensure it is validated and sanitized to prevent any potential security risks such as Remote Code Execution due to command injections.
2. Ensure permissions for the dnsmasq.conf file are secure and that only the necessary identities have write and read permissions.
3. The function is hardcoding the file paths. This can be enhanced by providing the flexibility to configure file paths through function parameters or configuration files.
4. Consider improving error handling so the function doesn't just exit, but maybe return a status or an error code or throw an exception, depending on the rest of your script.
5. You should wrap every variable in double quotes to prevent word splitting and pathname expansion. Although most of these variables seem safe, as their values are being provided by other functions, this function can be more robust and secure if it assumes nothing about the returned values.
6. Make sure that log files (`hps_log`) are secured and rotated on a regular basis to avoid any potential data leakage.

2.0.168 `create_config_dnsmasq`

Contained in `lib/functions.d/create_config_dnsmasq.sh`

Function signature: e8e5c3ae68c6edbb046d7d24d90b6f00e97a7553b5f4271d29d93e11655d696a

2.0.169 Function overview

The function `create_config_dnsmasq` is utilized for setting up and configuring the dnsmasq service. This function generates dnsmasq configuration about System, DHCP, DNS, TFTP and PXE by defining specific parameters obtained from the cluster configuration. It's important to note that the function utilizes a variety of global variables that are assumed to be pre-defined before the function runs. The function is invoked without any arguments.

2.0.170 Technical description

- **name:** `create_config_dnsmasq`
- **description:** This function sets up the dnsmasq service by generating a configuration file.
- **globals:**
  - `DHCP_IP`: Description (assumed to be IP address for DHCP)
  - `DHCP_IFACE`: Description (assumed to be network interface for DHCP)
  - `NETWORK_CIDR`: Description (assumed to be network CIDR for DHCP)
  - `DHCP_RANGESIZE`: Description (assumed to be DHCP's range size)
  - `DNS_DOMAIN`: Description (assumed to be the domain for DNS)
  - `HPS_TFTP_DIR`: Description (assumed to be the directory for TFTP)
- **arguments:** None.
- **outputs:** A dnsmasq configuration file.
- **returns:** Nothing explicitly but presumably exits the script if DHCP_IP global variable is undefined.
- **example usage:** `create_config_dnsmasq`

2.0.171 Quality and security recommendations

1. It's recommended to have error checking for all global variables used in this function to avoid misbehavior in case any of them is not defined.
2. Log all function execution and exit paths for debugging purposes and maintaining a traceable log of all actions taken by the function.
3. Check the result of the `cat` command for successful file creation and write in addition to the existence of the `DHCP_ADDRESSES` and `DNS_HOSTS` files.
4. Global variable names should be more descriptive to provide context about what value they should hold.
5. Avoid using `exit` function directly in the function; instead, return a status code and handle it in the caller function. Doing so allows for better error handling and script execution control.

2.0.172 `create_config_nginx`

Contained in `lib/functions.d/create-config-nginx.sh`

Function signature: b7278dfa14d865f0725cb56191fcc50aadd7039d4a7b5eaf44665de2c2e75027

2.0.173 Function overview

The `create_config_nginx` function is used to create a standard NGINX configuration specifically for the active cluster in a running environment. The function first identifies the active cluster file using the `get_active_cluster_filename` method. It then computes the path where the NGINX configurations will be stored by calling

the `get_path_cluster_services_dir` method. Finally, the function writes the NGINX configuration to the computed path.

2.0.174 Technical description

- **Function Name:** `create_config_nginx`
- **Description:** The function configures NGINX for the active cluster in a running environment. It finds the active cluster file, computes the path to store the NGINX configuration, and writes the configuration to this path.
- **Globals:** `NGINX_CONF: The variable stores the path where the NGINX configurations will be written.`
- **Arguments:** None
- **Outputs:** A file `"${NGINX_CONF}"` with NGINX configuration. Logs an info message that the nginx is being configured.
- **Returns:** No explicit return value.
- **Example Usage:** `bash create_config_nginx`

2.0.175 Quality and Security Recommendations

1. Check if the `get_active_cluster_filename` and `get_path_cluster_services_dir` methods correctly produce a result and handle any failure cases.
2. Define the 'worker_processes' and 'worker_connections' as variables at the top of your script to make them explicitly configurable.
3. Ensure proper permissions and ownership for the NGINX configuration file to avoid unauthorized access.
4. Validate and sanitize inputs to the function (if any will be added in the future) to prevent script injection attacks.
5. Consider implementing a backup mechanism for the existing NGINX configuration file before overwriting it.
6. Employ a script linter to enforce bash best practices and make the code more reliable and maintainable.

2.0.176 `create_config_nginx`

Contained in `lib/functions.d/create-config-nginx.sh`

Function signature: b7278dfa14d865f0725cb56191fcc50aadd7039d4a7b5eaf44665de2c2e75027

2.0.177 Function overview

The `create_config_nginx` function is utilized to create a new configuration for the nginx server. It sources the return value of the `get_active_cluster_filename` function, which should ideally refer to the active cluster filename. After defining the path

to the nginx configuration file, it logs that the nginx configuration is being set up. The function then creates a new nginx configuration file with predefined values.

2.0.178 Technical description

Name: `create_config_nginx`

Description: The function creates a new nginx configuration using predefined values. The resulting configuration file path is derived from the `get_path_cluster_services_dir` function which is appended with "/nginx.conf".

Globals: [`get_active_cluster_filename`: Provides the active cluster filename, `get_path_cluster_services_dir`: Provides base directory path, `hps_log`: logs the information]

Arguments: [None]

Outputs: This function outputs an nginx configuration file at the path defined by `NGINX_CONF`. The created file includes predefined configuration values.

Returns: There is no specific return value as the function directly operates and creates the nginx configuration file.

Example usage:

```
create_config_nginx
```

The example above will create an nginx configuration file with predefined values.

2.0.179 Quality and security recommendations

1. Perform checks: Before sourcing `get_active_cluster_filename`, it's necessary to validate that the function returns a valid, expected file path.
2. Error Handling: There should be an error-handling mechanism in place if `get_active_cluster_filename` or `get_path_cluster_services_dir` returns unexpected output.
3. Ensure Privacy: The path stored in `NGINX_CONF` should be protected to prevent unauthorized access.
4. Check for overwriting: Before overwriting the nginx configuration file, check that it does not already contain custom settings or sensitive data.
5. Code Readability: The use of global variables reduces code readability as it introduces dependencies on code outside the function. Consider defining them locally or passing them as input arguments.
6. Use safe `cat` redirect: There are potential risks with overwriting the nginx configuration file using `cat` with > redirect. It could be safer and recommended to use the -n option if available to prevent overwriting.

2.0.180 `create_config_rsyslog`

Contained in `lib/functions.d/create-config-rsyslog.sh`

Function signature: a86a9ea0160dcd4a1fe1c0a4875b3f38284da43ad49791bfd5e23ac15e63f10a

2.0.181 Function overview

The `create_config_rsyslog` function is used to create and set up a configuration file for the `rsyslog` service. It creates the required directories and log files, and sets global directives. It also loads all required modules for `rsyslog`, sets up UDP and TCP syslog reception, sets up the log format for various applications, and routes logs to specific files based on the program name.

2.0.182 Technical description

- **Name:** `create_config_rsyslog`
- **Description:** This function sets up `rsyslog` to perform logging operations for the system. It creates necessary directories and a configuration file, then populates this configuration file with directives that control `rsyslog` operations.
- **Globals:**
  - `RSYSLOG_CONF`: Path to the cluster services directory's rsyslog configuration file.
  - `RSYSLOG_LOG_DIR`: Path to the directory that contains rsyslog logs.
  - `HPS_LOG_DIR`: Path to the directory that contains HPS logs.
- **Arguments:** None
- **Outputs:** Writes to the RSYSLOG_CONF file.
- **Returns:** Nothing
- **Example usage:** `create_config_rsyslog`

2.0.183 Quality and security recommendations

1. Use more descriptive variable names for better code readability.
2. Sanitize input data to avoid risk of command injection.
3. Perform error handling and check if mandatory directories/files exist before processing.
4. Always use absolute directory paths to avoid ambiguity.
5. Set permissions based on the principle of least privilege.
6. Enclose variable references in double-quotes to protect them from word splitting and pathname expansion.

2.0.184 `create_config_rsyslog`

Contained in `lib/functions.d/create-config-rsyslog.sh`

Function signature: a86a9ea0160dcd4a1fe1c0a4875b3f38284da43ad49791bfd5e23ac15e63f10a

2.0.185 Function Overview

The `create_config_rsyslog` function is designed to create a rsyslog configuration in the cluster services directory with a pre-defined logging setup. This setup includes the creation of a directory for rsyslog logs, loading of necessary modules, defining global directives, creating specific templates for logging format, and routing logs based on specific conditions.

2.0.186 Technical Description

- **name:** create_config_rsyslog
- **description:** This Bash function creates a rsyslog configuration file in the cluster services directory with a specific logging setup.
- **globals:**
  - `RSYSLOG_CONF`: This global variable holds the full path to the rsyslog configuration file.
  - `RSYSLOG_LOG_DIR`: This global contains the path to the log directory for rsyslog.
- **arguments:** The function does not take any argument.
- **outputs:** The function outputs a created rsyslog configuration file with specific global directives and templates in place. It will also output a directory to store the rsyslog logs.
- **returns:** Not applicable.
- **example usage:**

  ```
  create_config_rsyslog
  ```

2.0.187 Quality and Security Recommendations

1. Error Handling: To improve the function quality, include error handling mechanisms to avoid potential failures during the directory creation or configuration file setup.
2. Permission Check: Check permissions before trying to create directories and files to avoid permission denied errors.
3. Logging: Improve logging by providing more detailed logs about what the function is doing at each step.
4. Validation: Perform validation on the path variables to ensure they are correctly set before proceeding with directory or file creation.
5. Security: Ensure the created directories and files have appropriate permissions to avoid unauthorized access.

2.0.188 `detect_call_context`

Contained in `lib/functions.d/system-functions.sh`

Function signature: f167df149452fd670d9f40bd87d52442f2f5d5153026bdfcab5f9c60997d7f96

2.0.189 Function Overview

The bash function `detect_call_context` is designed to identify and echo the current context in which the script is being executed. It handles three main contexts: when the script is sourced instead of directly executed, when it gets invoked as a common gateway interface (CGI), and finally, when it gets directly executed in a shell or reading from stdin without CGI environment variables. If none of these contexts apply, the function defaults to "SCRIPT".

2.0.190 Technical Description

- **Name:** `detect_call_context`
- **Description:** This function identifies the context in which the script is running, which could be either "SOURCED", "CGI", or "SCRIPT". It prints out the current context and then returns.
- **Globals:** [BASH_SOURCE[0]: Describes the source of the bash script, GATEWAY_INTERFACE: A required variable to detect CGI, REQUEST_METHOD: Another required variable to detect CGI, PS1: Helps in explicitly detecting the script]
- **Arguments:** None
- **Outputs:** Prints one of the four possible states - "SOURCED", "CGI", "SCRIPT", or a fallback "SCRIPT".
- **Returns:** `null`
- **Example Usage:**

  ```
  source your_script.sh
  detect_call_context
  ```

  This script would output "SOURCED" if `your_script.sh` contains a call to this function.

2.0.191 Quality and Security Recommendations

1. It is essential that global variables are well-defined and adequately protected in a function. Use local variables or provide default values to prevent empty or undefined global variables issues.

2. Bash lacks some advanced features like well-defined namespaces, classes, or functions. For better security and efficiency, consider using a more powerful scripting language like Python or Perl for complex scripts.
3. Make sure that the script running the function has appropriate permissions. Bash scripts can be a significant security risk if they are writable by any user. Therefore, secure your script by limiting access.
4. Implement error handling and fallbacks for unexpected behaviour or exceptions.
5. The function implicitly trusts that certain global environment variables are not maliciously set. Ensure that these environment variables are validated before use.
6. Regularly update your system and software to prevent security vulnerabilities.

2.0.192 `detect_client_type`

Contained in `lib/functions.d/network-functions.sh`

Function signature: a5996dd77379049ae4564d5c7eaab09a5189d239c610eea12c0d452cd96097e7

2.0.193 Function Overview

The function `detect_client_type()` is designed to determine the type of client making a request and echo it back. The function looks at both the query string and the user agent to make this determination. If the client type cannot be determined, it echoes back “unknown”.

2.0.194 Technical Description

- **Name:** `detect_client_type()`
- **Description:** Determines the type of the client from `$QUERY_STRING` or `$HTTP_USER_AGENT`. This function echoes the client type: ‘ipxe’, ‘cli’, ‘browser’, ‘script’ or ‘unknown’ if it can’t determine the client type.
- **Globals:**
  - `QUERY_STRING`: Utilized to determine the client type based on the presence of certain keywords. If not set, default value is an empty string.
  - `HTTP_USER_AGENT`: Utilized to determine the client type based on the presence of certain keywords. If not set, default value is an empty string.
- **Arguments:** The function does not accept any arguments.
- **Outputs:** Echoes the type of client making the request.
- **Returns:** Always returns 0 as the function is designed to not fail, falling back to a default output of “unknown” when necessary.
- **Example Usage:**

```
$ export QUERY_STRING="via=cli"
$ detect_client_type
cli
```

2.0.195 Quality and Security Recommendations

1. To reduce potential errors or misuse, explicitly document the environments and contexts in which this function should be used or not used.
2. Consider adding error handling for undesired or unexpected input to improve stability.
3. Make sure to sanitize user-generated inputs such as query strings to prevent injection attacks.
4. If feasible, add type checks for input values in cases where the function starts accepting arguments.
5. To prevent leakage of potentially sensitive information, use discretion with echoing data, especially if it includes data from HTTP headers in a web server environment.

2.0.196 `detect_storage_devices`

Contained in `lib/functions.d/storage_functions.sh`

Function signature: 26407abf67962b945819ec70c2d91c7a26b60c144eeb209e22298b013ba307ed

2.0.197 Function Overview

The function `detect_storage_devices` is used to identify all available block devices on a system and gather important details such as device model, vendor, serial number, type, bus, size, usage, and speed. This information is collected in a structured format for easy analysis and troubleshooting.

2.0.198 Technical Description

- **Name**: `detect_storage_devices`
- **Description**: This function detects all block devices in a Linux system and retrieves information of each device including the device path, model, vendor, serial number, bus type, device type, size, usage, and speed.
- **Globals**: None
- **Arguments**: None
- **Outputs**: The function generates a formatted string containing details about all detected storage devices.
- **Returns**: The function doesn't return a specific result – it echoes the output directly, making the output available in the standard output stream.
- **Example Usage** `detect_storage_devices` The function will deliver an output listing all the storage devices and their relevant details.

2.0.199 Quality and Security Recommendations

1. Always sanitize input, if any, to prevent any potential code injection attacks.

2. Incorporate error handling to make the function more robust. This can include scenarios wherein the queried device information is not available.
3. Develop a unit test case for the function to ensure its accuracy and validity.
4. Avoid potential command injection by checking names of the block devices before executing commands.
5. Assign meaningful names to the variable to make the code more readable.
6. Document the function usage and its arguments properly for clarity and future reference. If the function's behavior changes, update the documentation timely.
7. Instead of directly accessing hardware related information, consider using system APIs or other safer methods, if available.
8. The function currently prints information to standard output (via `echo`). Instead, consider returning status codes indicating success or failure for greater control over function reactions to specific scenarios.

2.0.200 `dns_host_add`

Contained in `lib/functions.d/dns-dhcp-functions.sh`

Function signature: ccef96f9190f5490f69c1c30476adc8c567555c449b389bdc1e3da826ee3a1ed

2.0.201 Function overview

The `dns_host_add` function is a bash function that adds or updates entries in the DNS hosts file. It accepts IP address, hostname, domain (optional), and alias (optional) as arguments. The function starts by validating the provided IP address and hostname, then it prepares the entry and checks for existing entries with the specified IP or hostname in the DNS host file. If an entry already exists, it is replaced by the new one; otherwise the new entry is added. After successfully manipulating the DNS host file, the function sends a HUP signal to the DNS caching daemon (dnsmasq) to force it to refresh the cache and then logs the completion of the process.

2.0.202 Technical description

- **Function Name:** `dns_host_add`
- **Description:** The function adds or updates a DNS host entry with given IP address, hostname and domain (optional). If the entry already exists in the DNS hosts file, it gets updated, otherwise it gets appended.
- **Globals:** `dns_hosts_file: It refers to the DNS hosts file.`
- **Arguments:**
  - `$1: The IP address for the new DNS host entry.`
  - `$2: The hostname for the new DNS entry.`
  - `$3 (Optional): The domain for the new DNS entry.`
- **Outputs:** Writes to the DNS host file.

- **Returns:** `1` for validation failures or creating/writing to file failures, `0` upon successfully adding or updating the DNS host entry.
- **Example usage:**

```
dns_host_add 192.168.1.1 host_name 'domain.com' 'alias.com'
```

2.0.203 Quality and security recommendations

1. Implement comprehensive error handling for each step of the process.
2. Log specific error messages whenever a validation check fails to aid in debugging.
3. Where possible, split the function into smaller subfunctions, each handling a specific task such as validation, file manipulation, logging etc. This enhances maintainability.
4. Implement file locks whenever manipulating the DNS hosts file to prevent race conditions.
5. Use secure temporary files when creating new ones to prevent symlink attacks.
6. Check that mandatory parameters are provided before proceeding with the function to prevent potential script breakage.
7. Store sensitive information like IP addresses securely and ensure logging does not reveal sensitive information.

2.0.204 `dns_host_get`

Contained in `lib/functions.d/dns-dhcp-functions.sh`

Function signature: 1c8acff7e11ab9876b1c501573a47211e80309f15d4b378face30b90d93c0e24

2.0.205 Function Overview

The function `dns_host_get()` is designed to retrieve information about a given host from a DNS hosts file within a specific cluster's services directory. It accepts one argument: a host identifier. The function reads from the DNS hosts file line by line, skipping over comments and empty lines. If the identifier matches an IP or a hostname in a line read from the file, it will print the entire line and cease operation, returning a success status. If no match is found, the function will return a failure status.

2.0.206 Technical Description

- Name: dns_host_get
- Description: This function retrieves a host's information by reading from a DNS hosts file in a cluster's services directory. It can search for either an IP or hostname provided through the identifier argument.
- Globals: [dns_hosts_file: The path to the dns_hosts file within the cluster's services directory]
- Arguments: [$1: The identifier, which can be either an IP or hostname]

- Outputs: Prints the line from the dns_hosts file that matches the identifier.
- Returns: This function returns 1 indicating failure if the dns_hosts file does not exist or if no matches for the identifier are found. If a match is found, the function returns 0 signifying a success.
- Example Usage:

```
dns_host_get "192.168.1.1"
dns_host_get "myhostname"
```

2.0.207 Quality and Security Recommendations

1. Implement stricter input validation: For instance, you could add more checks or regular expressions to validate IP or hostname format.
2. Handle errors and exceptions more granularly: Currently if the hosts file doesn't exist or is empty, the function will return 1 and cease operation. More informative error messages could be useful.
3. Be mindful of where and how this function is used. If mishandled, revealing the entire line from the dns_hosts file could potentially leak sensitive information.
4. Consider implementing a safer way to read files: Using a `while IFS= read -r line` loop within a function has potential for errors and may be clumsy in larger scripts.

2.0.208 `dns_host_remove`

Contained in `lib/functions.d/dns-dhcp-functions.sh`

Function signature: db24aaf06fea9141fd43e36856311228886e038a584f84280e8a80d2a4f12014

2.0.209 Function overview

The `dns_host_remove` Bash function is designed to update a DNS hosts file by removing a given entry, based on an identifier. The identifier can be an IP address or hostname. If the identifier is absent, or the DNS hosts file doesn't exist, an error or debug log is produced respectively. If a line in the DNS hosts file matches the identifier, that line is skipped (hence removed in the updated file). After removal, the function concludes by replacing the old DNS hosts file with the updated temporary file, whilst handling potential write failures.

2.0.210 Technical description

Function Definition Block:

- **Name:** `dns_host_remove`
- **Description:** Removes a host entry, specified by an identifier, from a DNS hosts file.

- **Globals:** None
- **Arguments:**
  - $1: Identifier, which could be the hostname or IP address to be removed from the DNS hosts file
- **Outputs:** Logs various statuses (error, debug, info) throughout the function execution.
- **Returns:** 1 if an error occurred (no hostname/IP provided or failed to write DNS hosts file); 0 under normal operation.
- **Example Usage:** `bash dns_host_remove "localhost"` *This will remove all lines with "localhost" from the DNS hosts file within the cluster services directory.*

2.0.211 Quality and security recommendations

1. Validation of inputs: Input should be validated to ensure that it complies with expected formats. For example, if an IP address is expected, the function should check if the input is a valid IP address.
2. Error Messages: Error messages should not reveal too much information about the internal structure of the function/script to avoid potential leaks of sensitive information.
3. Error handling: The function should not only return an exit code, but also handle the error internally. This could be done by implementing fallback strategies, indicating where the error happened or by providing more context.
4. Atomicity: The script attempts to achieve atomicity by replacing the old file with a new one. However, consider scenarios where write operations may fail or be interrupted.
5. Temp File handling: Temporary file cleanup is useful to avoid unnecessary disk space consumption and potential sensitive data leakage.
6. Use of functions like "hps_log": Ensure that these function calls do not pose a security risk via command injection or other vulnerabilities. They should also handle internal errors gracefully.

2.0.212 `_do_pxe_boot`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 34359020b853100f39729c1ec34d66fe43cbe72d4231aa30342ee1342e7faf69

2.0.213 Function Overview

The bash function `_do_pxe_boot()` is designed to implement a PXE (Preboot Execution Environment) boot using a given kernel and initrd (initial RAM disk). The function takes two arguments - `kernel` and `initrd`, which are expected to be paths to the necessary files. The function validates these arguments, logs the operation details, and executes the PXE boot sequence.

2.0.214 Technical Description

- **Name**: `_do_pxe_boot()`
- **Description**: Performs a PXE boot using the provided kernel and initrd. If the kernel or initrd is not provided, the function will log an error and exit.
- **Globals**: `IPXE_BOOT_INSTALL` - stores a generated iPXE boot script.
- **Arguments**:
  - `$1: kernel` - The path to the kernel file needed for the boot.
  - `$2: initrd` - The path to the initial ramdisk (initrd).
- **Outputs**: Logs various messages and outputs the boot installation script.
- **Returns**: 1, if either the kernel or initrd parameter is not provided, else returns nothing.
- **Example Usage**: `_do_pxe_boot "/path/to/kernel" "/path/to/initrd"`

2.0.215 Quality and Security Recommendations

1. Ensure the function handles other potential errors, such as if the provided paths do not exist or point to invalid files.
2. Instead of using a global variable `IPXE_BOOT_INSTALL`, consider returning the boot installation script directly, making the function more reusable.
3. Consider implementing a mechanism that extracts and checks the format or validity of the kernel and initrd files.
4. Encrypt sensitive logs or do not log sensitive messages that might expose vulnerable system details.
5. Validate the input arguments beyond just checking if they're non-empty. For instance, verify they're valid file paths for security and robustness.
6. Update the code comments to be more explicit about the actions being performed for future code maintenance and understanding.

2.0.216 `_do_pxe_boot`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 34359020b853100f39729c1ec34d66fe43cbe72d4231aa30342ee1342e7faf69

2.0.217 Function Overview

The function `_do_pxe_boot` is a bash function that takes two arguments, a kernel and an initrd (initial RAM disk). It checks if these parameters are non-empty and logs an error if they are. If everything is in order, it logs some debug information, creates an iPXE boot install script with the given parameters and a datestamp, and then outputs this script.

2.0.218 Technical description

- **name**: `_do_pxe_boot`
- **description**: Boots the system over a network via iPXE method with the provided kernel and initrd. It generates iPXE boot commands on the fly and outputs them for further usage.
- **globals**: [IPXE_BOOT_INSTALL: stores the boot script for iPXE]
- **arguments**: [$1: The specified kernel to boot, $2: The initial RAM disk to be used while booting]
- **outputs**: The iPXE boot script as printed to the stdin.
- **returns**: Error state 1 if either a kernel or an initrd were not given. Else nothing is returned.
- **example usage**:

```
_do_pxe_boot "vmlinuz" "initrd.img"
```

2.0.219 Quality and Security Recommendations

1. Validate the arguments not only for non-emptiness but also for their correct format and for being legitimate files existing in the system.
2. Consider utilizing more distinctive log levels during logging operations for more precise troubleshooting.
3. Through incorporating error handling mechanisms, ensure that the function behaves properly even under unexpected conditions.
4. Always quote the variables in bash to avoid word splitting and pathname expansion.
5. Avoid the usage of uppercase for non-global and non-readonly variables to prevent name clashes with shell variables.

2.0.220 `download_alpine_release`

Contained in `lib/functions.d/tch-build.sh`

Function signature: f1438a9265c0b60b2947c704f30ee5980245150741b8578a6cb28e185b7cd407

2.0.221 Function Overview

The bash function `download_alpine_release()` aims to download a specific version of the Alpine Linux distribution. If no version is specified, it will try to determine and download the latest available version. The specific requirements of this function include a specified Alpine version input and a path location under the HPS_RESOURCES environment variable where the downloaded file will be stored.

2.0.222 Technical Description

- **Name**: `download_alpine_release()`

- **Description**: This shell function is designed to download the specified version of the Alpine iso and save it under a defined path. If no version is specified, it downloads the latest version. If the file already exists in destination path, it does not download but returns the file path.
- **Globals**:
  - HPS_RESOURCES: The destination directory for the downloaded Alpine ISO. If not set, function will return error.
- **Arguments**:
  - `$1` (optional): Version of the Alpine Linux distribution to download.
- **Outputs**:
  - Outputs log messages via `hps_log()` function.
  - Prints the file path of the downloaded ISO.
- **Returns**:
  - `0` for successful downloads or if the file is already available.
  - `1` for missing HPS_RESOURCES or for failure in detecting the latest Alpine version.
  - `2` for failure in downloading the file.
- **Example Usage**:

```
download_alpine_release 3.20.2
```

2.0.223 Quality and Security Recommendations

1. Add input validation to ensure correct format of the `alpine_version` if provided.
2. Introduce a verbose mode to provide additional information during the download process.
3. Implement checksum validation after download to ensure the integrity of the downloaded iso.
4. Make the function more general by allowing the user to define which architecture (x86_64, armv7, etc.) they would like to download instead of always downloading the x86_64 version.
5. The HPS_RESOURCES variable should be verified not only as a non-empty string but also as a valid writable directory.

2.0.224 `download_file`

Contained in `lib/functions.d/system-functions.sh`

Function signature: 77275d6db2a730d2f09e1f7111242d9607b215be97269a81302557f4e047a820

2.0.225 Function Overview

`download_file` is a Bash function designed to download a file from the provided URL to the specified destination path. The function uses either `curl` or `wget` based on availability and supports resuming interrupted downloads. If provided, the function verifies the checksum of the downloaded file using `sha256sum`. If an error is encountered (missing arguments, failure to create the destination directory, download failure, or checksum mismatch), the function will log the error and return a non-zero exit code.

2.0.226 Technical Description

- **Name:** `download_file`
- **Description:** Bash function to download a file from a URL to a specified destination path, with optional SHA256 checksum verification.
- **Globals:** None.
- **Arguments:**
  - `$1: url` - The URL of the file to be downloaded.
  - `$2: dest_path` - The destination path where the downloaded file will be placed.
  - `$3: expected_sha256` - (Optional) The expected SHA256 checksum of the downloaded file.
- **Outputs:** Information, warning and error logs.
- **Returns:** 0 if the file is successfully downloaded and the checksum (if provided) matches. 1 if necessary tools are missing or required arguments are not provided, 2 if any operation (e.g., directory creation or file download) fails, and 3 if the checksum does not match.
- **Example Usage:**

```
download_file "https://example.com/test.txt" "/path/to/test.txt"
 ↪  "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
```

2.0.227 Quality and Security Recommendations

1. The `download_file` function currently lacks input validation. Malformed or malicious values for the URL or destination path could lead to unexpected behavior or security risks.
2. The function silently falls back to `wget` if `curl` is not available. Explicitly specifying the desired tool or alerting the user to the fallback could improve transparency and control.
3. If `sha256sum` is not available, the function logs a warning but continues the download. It might be preferable to fail outright to support secure environments where checksum verification is required.
4. The function only supports SHA256 for checksums. Supporting additional or newer checksum methods could improve versatility and security.

5. It may be worth considering a timeout for the download operation, to prevent hanging in the case of a slow or unresponsive server.

2.0.228 `download_iso`

Contained in `lib/functions.d/iso-functions.sh`

Function signature: 8809a8fec0ae6a5ffcfad851cf4f532fecb9538381ebc77a93c97b6f36151885

2.0.229 Function overview

The `download_iso` function is designed to download ISO files based on specific parameters given as inputs. The parameters include central processing unit (CPU) type, manufacturer (MFR), operating system name (OS name), and operating system version (OS version). The function utilizes a helper function to get the directory path where the ISO will be saved. The function downloads the ISO file from the appropriate URL based on the parameters and saves it to the constructed path. If the ISO file is present, the function will not download it. The function handles exceptions such as failure in downloading, unsupported OS variant, etc., and it further communicates its success or failure through echo statements.

2.0.230 Technical description

- **Name:** download_iso
- **Description:** Downloads specified ISO image file from the internet, using curl, and saves it in mentioned directory.
- **Globals:** None
- **Arguments:**
  - $1: Central processing unit type.
  - $2: Manufacturer.
  - $3: Operating system name.
  - $4: Operating system version.
- **Outputs:** Echoes status of the operations performed and download link if successful.
- **Returns:** Status code of the command in case of failure or success in downloading ISO file.
- **Example usage:**

```
download_iso "x86_64" "Intel" "rockylinux" "10.0"
```

2.0.231 Quality and security recommendations

1. For better error handling, consider defining custom error messages, so that users can understand what type of error occurred instead of a simple failure message.

2. It would be a good idea to also clean up the partial downloads from failed curl attempts.
3. Always validate the inputs. For the current function, check if all the required arguments are provided and properly formatted.
4. You might consider catching more specific errors e.g, network errors, to give more detailed feedback to the user.
5. It's a good practice to ensure that the directory where the ISO is being saved has proper permissions to avoid any potential security risks.
6. Revealing the URLs or paths of the ISO files being downloaded in echo messages could be a security risk. Consider anonymizing or masking these details.

2.0.232 `ensure_opensvc_installed`

Contained in `lib/functions.d/opensvc-function-helpers.sh.sh`

Function signature: f18e6428a93c8ba30f5315e22c187c0ed72b465e9e6f3bd8d40a56ca8a934349

2.0.233 Function Overview

The bash function `ensure_opensvc_installed` is designed to ensure that OpenSVC is installed and operational on the system. To achieve this, it performs a series of checks and installations procedures: 1. It first checks if `om` command (a pivotal OpenSVC command) already exists and is executable. 2. If `om` is found, the function verifies that it indeed runs successfully. 3. If the `om` command is not found or is not functional, then the function attempts to install OpenSVC using the `ips_install_opensvc` function. 4. Finally, it verifies whether the installation was successful by checking for the 'om' command again and verifying its execution.

2.0.234 Technical Description

Following is the pandoc-safe block detailing function's technical specifics:

- **Name**: `ensure_opensvc_installed`
- **Description**: The function checks if OpenSVC is installed and if not, attempts to install it. It also makes sure that the installation is successful and the 'om' command is functional.
- **Globals**: None
- **Arguments**: None
- **Outputs**: Various log messages (info, debug, warning, error) depending upon the existence and functionality of OpenSVC.
- **Returns**: 0 if OpenSVC is installed and operational. 1 if OpenSVC is not installed, unable to install, or if the installation is successful but 'om' command is not functional.
- **Example Usage**:

```
ensure_opensvc_installed
if [ $? -eq 0 ]; then
  echo "OpenSVC is installed and operational."
else
  echo "OpenSVC installation failed or 'om' command not
   ↪  functional."
fi
```

2.0.235 Quality and Security Recommendations

1. Add more validation checks to make sure that all prerequisites for running and installing OpenSVC are met.
2. Implement more robust error handling mechanisms to better trace possible installation or operational errors.
3. Check for the successful execution of all the intermediary commands and not only the last command in each if-statement.
4. Enhance logging by appending logs to a file for post-run diagnostics.
5. Consider running some preliminary OpenSVC commands post-installation to ensure the proper functioning of OpenSVC, rather than solely relying upon the 'om' command.

2.0.236 `exec_wrapper`

Contained in `lib/functions.d/system-functions.sh`

Function signature: ed73acb7396b6167dad4a7946bf86389579d207e755df9082e74f576e3b01824

2.0.237 Function Overview

The `exec_wrapper()` function is a Bash function designed to execute commands and handle errors elegantly. It captures the standard error (stderr) from the command execution into a temporary file. If the command executes successfully, it cleans up the temporary file and finishes execution. However, if the command fails (indicated by a non-zero exit code), it logs the failed command, the exit code, and any content in the standard error. Once it finishes error handling, it deletes the temporary file and returns the original command's exit code.

2.0.238 Technical Description

- **Name:** `exec_wrapper()`
- **Description:** It's a Bash function for executing commands. Records stderr into a temporary file and provides an error logging mechanism for any non-successful exit codes (non-zero). It then cleans up the temporary files and returns the original command's exit code.

- **Globals:** None
- **Arguments:** `[$1: Command to be executed]`
- **Outputs:** Logs error messages in case the command execution fails.
- **Returns:** The original command's exit code.
- **Example Usage:**

```
command="ls non_existent_directory"
exec_wrapper "$command"
```

2.0.239 Quality and Security Recommendations

1. Ensure all variables used in the function, like `cmd`, `stderr_file`, etc. are localized using the `local` keyword to prevent any side effects from global variables.
2. Implement thorough input validation for the `cmd` variable to avoid command injection vulnerabilities.
3. Consider implementing a mechanism to limit the maximum size of the `stderr_output` to prevent possible out-of-memory errors.
4. In the error logging situations, consider including more substantial and helpful messages to assist in troubleshooting.
5. Before deleting the stderr file, ensure the file exists to avoid unnecessary error messages. You may use the `-f` flag for the `rm` command to force deletion without warnings.
6. Enforce usage of this function within a single logical scope (like a single script or function) without any dependencies on other scopes or parent scopes to maintain good encapsulation.

2.0.240 `export_dynamic_paths`

Contained in `lib/functions.d/system-functions.sh`

Function signature: a6b2a47e08ed460524c491151fd944d515572f0fe9d26a1a2d5d310ad72b99b5

2.0.241 Function Overview

This Bash function, `export_dynamic_paths`, is designed to set and export paths dynamically within a cluster server environment. It makes use of local cluster names to base its operation while providing an alternative default directory path. This function is significant for managing multiple active clusters and ensuring that the active cluster is properly recognized. The function also sets and exports environment variables representing various paths in the cluster's configuration.

2.0.242 Technical Description

- **Name**: `export_dynamic_paths`

- **Description**: A Bash function designed to set and export cluster configuration paths dynamically using the provided cluster name as a reference. This includes the active cluster, the cluster's configuration directory, and the hosts configuration directory. The function also considers the case where an active cluster has not been specified.
- **Globals**: [`HPS_CLUSTER_CONFIG_BASE_DIR`: This global variable provides the root directory for storing cluster configs. By default, its value is set to /srv/hps-config/clusters]
- **Arguments**:
  - `$1`: This argument is the string value that represents the cluster name. If it is not provided, the function will use the currently active cluster (default to empty string).
- **Outputs**: Outputs a warning message "[x] No active cluster and none specified." to stderr if no active cluster exists and none is specified by user.
- **Returns**: Returns 1 if there is no active cluster and none has been specified by the user, or 0 if execution was successful.
- **Example usage**: `export_dynamic_paths 'cluster_name'`

2.0.243 Quality and Security Recommendations

1. Validate inputs at the start of the function. Be sure that the supplied cluster name does not contain unsafe characters (e.g., slashes, backticks, etc.) that could potentially lead to command or path injection attacks.
2. Handle all error or exceptional scenarios. Improve error handling so that more specific messages are returned based on the failure's nature.
3. Make sure that proper permissions are set for the directories and files involved, especially when the function is handling paths and using these to access potentially sensitive data or system configurations.
4. Incorporate a logging mechanism to trace the function's behavior when debugging is required to aid in future troubleshooting.
5. Use more descriptive variable names for readability and maintenance. Ensure the variable and function names accurately describe their purposes or behavior.

2.0.244 `extract_iso_for_pxe`

Contained in `lib/functions.d/iso-functions.sh`

Function signature: 01dad08860894440c0b140af9d5906479a477709165bc36b719668d2a79b04b3

2.0.245 Function Overview

The bash function `extract_iso_for_pxe` is designed to extract an ISO file for PXE (Preboot eXecution Environment) boot purposes. The function accepts specific hardware and operating system details as parameters and utilizes these details to find and extract

the appropriate ISO file from a predetermined directory. If an ISO file corresponding to the described parameters does not exist, the function will return an error.

2.0.246 Technical Description

- **Name**: `extract_iso_for_pxe`
- **Description**: This function is used to extract an ISO file for PXE boot. It does a verification if the required ISO is already extracted or not. If not, then it extracts the ISO to a specified directory.
- **Globals**: []
- **Arguments**:
  - `$1: cpu` - Represents the CPU details needed to find the correct ISO.
  - `$2: mfr` - Manufacturer details.
  - `$3: osname` - The name of the operating system.
  - `$4: osver` - The version of the operating system.
- **Outputs**: Outputs status and error messages to STDOUT, and error messages to STDERR.
- **Returns**:
  - 0 - On successful extraction or if the ISO file is already extracted.
  - 1 - When the required ISO is not found or if there is a failure in ISO extraction.
- **Example usage**: `extract_iso_for_pxe "$cpu" "$mfr" "$osname" "$osver"`

2.0.247 Quality and Security Recommendations

1. Always make sure to validate the values of the input parameters given the sensitive nature of 'extract_iso_for_pxe'. This function directly influences the OS being fetched and used and could lead to a compromised system if input isn't verified.
2. The function has no way of handling or sanitizing unexpected input. Adding input validation can prevent errors and possible security risks.
3. Take into consideration the error messages. They expose the directory structure to the STDERR, which could potentially be a security risk.
4. We could consider adding more error traps or signals to handle other potential issues like low disk space, permissions etc. during the extraction process.

2.0.248 `_extract_metadata_field`

Contained in `lib/functions.d/node-libraries-init.sh`

Function signature: 87fd013e60dcf62b09ec353157ed34c83ab7d30fb8d6476164a8d6ac45aa8da7

2.0.249 Function overview

The `_extract_metadata_field` function is used to capture specific data from a given metadata string. It takes two arguments: the metadata string and the specific field to be extracted from the string. Using a pattern, it captures a value from the given field (if available) and echoes out the value.

2.0.250 Technical description

Name: `_extract_metadata_field`

Description: The function `_extract_metadata_field` is used to extract the data for a particular field from a given metadata string. It does this by building a pattern that it then matches against the metadata string. If a match is found, it outputs the matched field's value.

Globals: None

Arguments:

- `$1`: This is the first argument passed to the function. It is the metadata string from which a field's value is to be extracted.
- `$2`: The second argument given to the function. This is the specific field that the function should try to extract a value for from the metadata string.

Outputs: If a match is found, the function outputs the value of the field that was matched in the metadata string.

Returns: The function doesn't have a specific return value. However, if the function successfully matches the pattern against the metadata string then it will echo the value back.

Example usage:

```
_extract_metadata_field "name=John age=23" "age"
# Outputs: 23
```

2.0.251 Quality and security recommendations

1. Always ensure to sanitize input before using inside a pattern matcher. This will prevent any potential issues from malformed or malicious input strings.
2. It is highly advisable to handle the case of no match being found. Recently, the function does not produce any output or warnings in case the field does not exist in the input metadata string.
3. The function can be modified to return an error code if a match is not found, which will further allow better error handling for the caller.
4. This function assumes that the field values will not contain spaces. If spaces are a possibility in your use case, further potential improvements can be made to handle such cases.

2.0.252 `extract_rocky_iso_for_pxe`

Contained in `lib/functions.d/iso-functions.sh`

Function signature: 58aa7f0eeaccca253f8f5e0ac76c7e61b0821f7025275c788df4e2053ce48fec

2.0.253 Function overview

The `extract_rocky_iso_for_pxe` function is designed to extract the Rocky Linux ISO for PXE (Preboot eXecution Environment) broadcasts. This process involves taking the specified ISO file and distributing its contents to a directory labelled according to the architecture and manufacturer, while also incorporating the operating system's name and the provided version. Extraction methods include either using `bsdtar` or `fuseiso`, depending on which is available. If neither are found, the function will return an error.

2.0.254 Technical description

- **Name:** `extract_rocky_iso_for_pxe`
- **Description:** Extracts a specified Rocky Linux ISO file for PXE broadcasting.
- **Globals:**
  - `MFR`: Describes the manufacturer of the Linux system
  - `OSNAME`: Specifies the name of the operating system
- **Arguments:**
  - `$1`: Path of the ISO file to extract
  - `$2`: Version of the Rocky Linux OS
  - `$3`: CPU architecture
- **Outputs:** Extracts the ISO file content and reports the errors (if any) and completion messages.
- **Returns:** The function returns `1` if neither `bsdtar` nor `fuseiso` are found in the system.
- **Example usage:**

```
extract_rocky_iso_for_pxe "./rocky.iso" "8.4" "x86_64"
```

2.0.255 Quality and security recommendations

1. File path validation: The function could include validation to check if the provided ISO file path actually exists and is readable. This would help avoid unnecessary function failures and provide a more user-friendly error message.
2. Version validation: The function could also include validation to ensure that the provided OS version matches the expected format.
3. CPU type validation: To avoid potential issues, the function could validate that the provided CPU type is supported.
4. Security enhancements: The use of `mktemp` and `fuseiso` could potentially be exploited if not properly secured. It would be worth looking into ways to

better secure these operations by, for example, limiting permissions and regularly reviewing and updating the function to respond to known vulnerabilities.

5. Error handling improvements: More comprehensive error handling could be implemented to catch, interpret, and handle different error cases more effectively.

2.0.256 `fetch_and_register_source_file`

Contained in `lib/functions.d/prepare-external-deps.sh`

Function signature: 97650ca173393457a3e6151b2fbb0a997e425c3010fc3c13018364e050618d70

2.0.257 Function Overview

The function `fetch_and_register_source_file()` is designed to first download a file from a specified url and then register this file using a given handler. The function accepts three arguments; the url from which the file will be downloaded, the handler which will be used to process the file after download and the filename of the downloaded file. If the filename is not provided, the function will take the base name from the url.

2.0.258 Technical Description

- Name: `fetch_and_register_source_file`
- Description: This function is used to download a file from a given url and then register it using a provided handler.
- Globals: None.
- Arguments:
  - $1: url from which the file will be downloaded.
  - $2: handler used to process the file after the download.
  - $3: filename of the downloaded file. If not provided, the function will derive it from the given url.
- Outputs: The function performs the task of downloading and registering a file. There are no specific output returns on the console.
- Returns: The function will return the status of the last command executed within it. Thus, if both the fetch and register operations are successful, the function will return `0`. If either operation fails, the function will return the corresponding error status.
- Example usage:

```
fetch_and_register_source_file "http://example.com/file.txt"
 ↪  "myHandler"
```

2.0.259 Quality and Security Recommendations

1. Validate inputs: For robustness and security, validate the input parameters to ensure they are in the correct format and have valid values.

2. Handle download issues: Consider adding more granular error handling for the `fetch_source_file` function to allow the function to easily troubleshoot issues related to file download.
3. Handle registration issues: Similarly, address possible failure points in `register_source_file` with adequate error handling and messaging.
4. Check handler validity: Before invoking the handler on the downloaded file, ascertain if the handler exists and can be executed safely.

2.0.260 `fetch_source_file`

Contained in `lib/functions.d/prepare-external-deps.sh`

Function signature: 9851dec43ce9f3e94856800874741f8ff28deda9346709daaa88282763e49999

2.0.261 Function overview

The `fetch_source_file` function is a utility to download a file from a given URL and save it on a specified destination directory. It first checks if the file already exists in the destination directory. If the file does not exist, it downloads the file using `curl` command. If the `filename` argument is not provided, it infers the filename from the URL. The default directory to save the file is `/srv/hps-resources/packages/src`, nevertheless, it can be overwritten through `HPS_PACKAGES_DIR` environment variable.

2.0.262 Technical description

- **Name**: `fetch_source_file`
- **Description**: Fetches a file from a provided URL and saves it in a specific location.
- **Globals**: [`HPS_PACKAGES_DIR`: Specifies the root directory for saving the downloaded files]
- **Arguments**: [`$1`: URL of the file to be downloaded, `$2`: Name of the downloaded file]
- **Outputs**: Logs to stdout showing the progress and result of the download.
- **Returns**: `0` if the file is successfully downloaded or already exists on the server, `1` if the download fails.
- **Example usage**:

  ```
  fetch_source_file "https://example.com/file.zip" "myFile.zip"
  ```

2.0.263 Quality and security recommendations

1. Input validation: As the function downloads content from a URL, it is advisable to add checks for ensuring that the URL is properly formatted and secure (uses

`https://`, belongs to trusted domain etc).

2. Error handling: While the function checks if the file is successfully downloaded, it would be better to add error handling for other operations as well like creating directory.
3. Sanity checks: It would be safer to add some sanity checks on the downloaded file, like checking its size, verifying its checksum and more.
4. Avoid using global variables: The use of global variables makes code hard to predict and debug, it is better to pass them as parameters to the functions.
5. Logging: consider redirecting error logs to stderr consistently. In the current case, some logs are written to stdout, some - to stderr, which might be cumbersome to debug if the function is used in a script.

2.0.264 `_find_all_inits`

Contained in `lib/functions.d/node-libraries-init.sh`

Function signature: 01aa7f30fc63f2659f28c6e3d25c403f8ba0e1c854c5bc7d5ea41b27e81d09f8

2.0.265 Function Overview

The bash function `_find_all_inits()` performs a search operation to find all .init files in specified directories. The directories it searches are determined based on the arguments provided to the function, with a base directory being passed as the first argument and the OS version as the second argument. The function then defines an array of search paths including the base directory and the version-specific directory of the OS. If these directories exist, it uses the `find` command to search for any .init files located within them. The search results are then sorted before being returned by the function.

2.0.266 Technical Description

- **Name**: `_find_all_inits`
- **Description**: This bash function searches for .init files within specified directories separated by base directory and OS version directory. It uses the `find` command to locate the files and the command `sort` to arrange the results in a sorted order.
- **Globals**: None
- **Arguments**:
  - `$1`: Base directory to search in (base_dir)
  - `$2`: OS version directory to search in (os_ver)
- **Outputs**: The paths of the found .init files, if any, sorted in lexicographical order
- **Returns**: No explicit return value. However, if successful, it will echo sorted list of the paths of the .init files found during the operation.
- **Example usage**: `_find_all_inits "/home/user" "v1.0"`

2.0.267 Quality and Security Recommendations

1. Always validate whether the provided input directories exist and handle the error properly if they do not.
2. Use secure methods to process paths and handle files to prevent potential arbitrary file read vulnerabilities.
3. Consider setting and enforcing a maximum depth for the `find` command to prevent potential denial-of-service if the directory structure is too deep.
4. Where critical, use tamper-evident logging that records what the script does, and not merely what it finds. This can help detect unauthorized changes later.
5. Always test the function in a controlled environment before deploying it in a production system.

2.0.268 `_find_available_hostname`

Contained in `lib/functions.d/host-functions.sh`

Function signature: e1ebf125bbaa40a57dbcf322796ec2cbea2e52343795101f43b0ed3b5626efa3

2.0.269 1. Function Overview

The function `_find_available_hostname` is utilized to find an available hostname from an existing set of hostnames for a given type. The function determines the highest number assigned to the existing hostnames and increments it by 1 to maintain uniqueness. Once the new number is determined, it appends it to the host type, ensures proper formatting and returns this value.

2.0.270 2. Technical Description

- **Name:** `_find_available_hostname`
- **Description:** A function that generates a unique hostname by incrementing the highest existing number in the set of hostnames for a given type. Returns the generated hostname.
- **Globals:** None
- **Arguments:**
  - `$1`: The type of the host. Will be converted to lowercase.
- **Outputs:** Prints the generated unique hostname, formatted as “hosttype-lowercase-{next_number filled to 3 places}”
- **Returns:** Returns 0 after successfully generating and printing the unique hostname.
- **Example Usage:** `_find_available_hostname Master`

2.0.271 3. Quality and Security Recommendations

1. Ensure that the argument passed to the function is always sanitized and validated to prevent potential security issues.
2. Currently, the function maintains its own error handling for invalid results. Consider centralizing error handling for consistency and ease of maintenance.
3. Consider adding checks in place to handle cases where the `get_cluster_host_hostnames` function returns an error, or more values than expected.
4. The regex matching operation might potentially turn into a time-consuming operation if dealing with a large set of hostnames. Optimization here could be considered.
5. Organize the function into smaller functions each handling single responsibility to improve readability and maintainability.
6. Consider using a more elaborate logging system to provide detailed logs for easier bug tracing and process understanding.
7. Remember to thoroughly test the function with large data sets to ensure it can handle larger load scenarios.

2.0.272 `_find_available_ip`

Contained in `lib/functions.d/host-functions.sh`

Function signature: f21c0add962ad9b971cdda7e66770a3e4925154b0c1a68a3eb1f09996cf1d293

2.0.273 Function overview

The `_find_available_ip` function is intended to find an available IP for a host in a cluster. It takes three arguments: the MAC address of the host (`mac`), the start IP of the DHCP server (`dhcp_ip`), and the range of IPs the DHCP server can assign (`dhcp_rangesize`). The function scans across the range of possible IPs to find one that is not currently in use by a host in the cluster.

2.0.274 Technical description

- **name**: `_find_available_ip`
- **description**: This function computes the available IP for a MAC address provided there is an available slot within the DHCP server range.
- **globals**: None
- **arguments**: [`$1`: MAC address of the host machine, `$2`: IP address of the DHCP (`dhcp_ip`), `$3`: size of the IP range that can be assigned (`dhcp_rangesize`)]
- **outputs**: Outputs an available IP address if one is found.
- **returns**: Returns 0 if an available IP was found; Returns 1 if no available IP was found.
- **example usage**: `_find_available_ip "00:0a:95:9d:68:16" "192.168.1.1" "254"`

2.0.275 Quality and security recommendations

1. Consider sanitizing the inputs to the function to protect against potential injection attacks.
2. Incorporate error handling mechanisms to handle unexpected function behavior such as inability to convert IP to integer or failing to get host configuration.
3. Implement check for extreme edge cases like negative range size and unformatted MAC or IP address.
4. Consider implementing function timeouts to prevent hung processes from halting the system.
5. Be sure to adequately secure the system which houses this data, as malicious access to MAC addresses and knowledge of their associated IP addresses can compromise the security of the network.

2.0.276 `find_hps_config`

Contained in `lib/functions.sh`

Function signature: b3ba10a8967a3088d5d8dae7f4bd970972f7563b406d9440431b21d23e4b538d

2.0.277 Function Overview

The function `find_hps_config` is used to find the configuration file for High-Performance Server (HPS). The locations to search for the file are contained in the array `HPS_CONFIG_LOCATIONS`. It iterates over each location in the array until it finds a non-empty file, and returns the path of this file. If no such file is found the function returns an error.

2.0.278 Technical Description

- **Name**: `find_hps_config`
- **Description**: The function scans for a configuration file in several predefined locations specified in `HPS_CONFIG_LOCATIONS`. Upon finding the configuration file, it outputs its location and terminates with a success status. If no configuration file is found, it returns an error status.
- **Globals**: `[ HPS_CONFIG_LOCATIONS: An Array containing the locations to search for the configuration file ]`
- **Arguments**: None
- **Outputs**: The file path of the located configuration file.
- **Returns**: `0` if it successfully locates the configuration file, `1` if it can't find any such file.
- **Example usage**: `config_location=$(find_hps_config)`

2.0.279 Quality and Security Recommendations

1. It's vital to verify permissions of the configuration file before reading it. An improperly protected file can be altered by malicious parties.
2. When `HPS_CONFIG_LOCATIONS` is being defined, ensure that the locations in this array are trusted sources to prevent configuration hijacking.
3. For improved security, consider implementing a validation of the content of the configuration file to ensure it's not corrupted or compromised.
4. Convey errors not just as return codes but also as output to `stderr` to help troubleshoot potential issues.

2.0.280 `format_mac_colons`

Contained in `lib/functions.d/network-functions.sh`

Function signature: 5496abb67b397a89b718680b3f650d5c789886634c83a14eef1474f51a2ddb06

2.0.281 Function Overview

The `format_mac_colons()` function takes in a MAC address as an argument and outputs the MAC address in lowercase with colons inserted after every two characters. It first validates whether the provided MAC address is valid and consists entirely of hex characters. If the MAC address is missing or in an invalid format, the function will output an error message to stderr and halt execution.

2.0.282 Technical Description

- **Name**: `format_mac_colons()`
- **Description**: This function formats a MAC address by removing any existing delimiters (colons, dashes, dots) and then re-inserting colons after every two characters. The outputted MAC address is always in lowercase.
- **Globals**: None
- **Arguments**:
  - `$1 (mac)`: the MAC address to be formatted
- **Outputs**: If the input MAC address is valid, it outputs the formmated MAC address to stdout. If the input MAC address is invalid or missing, it outputs an error message to stderr.
- **Returns**: The function returns 0 if the MAC address was successfully formatted. It returns 1 and ceases execution if the MAC address is invalid or missing.
- **Example usage**: `format_mac_colons "AB-CD-EF-12-34-56"`

2.0.283 Quality and Security Recommendations

1. **Error Checking**: Add explicit error checking for the input MAC address argument to ensure that it meets expected characteristics such as its length and whether it

only contains valid characters.

2. **Detailed Error Messages**: Include more details in the error messages for easier troubleshooting. For example, the error message can indicate which part of the MAC address string is incorrect.
3. **Testing**: Add more comprehensive unit tests for this function to cover all edge cases.
4. **Documentation**: Document the expected format of a MAC address explicitly at the start of the function.
5. **Robust Input Handling**: The current handling of different delimiter characters ('-', ':', '.') is good, but could be made more robust by adding support for variations in letter case (upper/lower) and spacing within the MAC address.

2.0.284 `generate_dhcp_range_simple`

Contained in `lib/functions.d/network-functions.sh`

Function signature: d040dcc216635e3158248e2fd6c7c0ba131ed32bec1b3b2c1708b14f4efd6311

2.0.285 Function overview

The `generate_dhcp_range_simple()` function computes the range of addresses to be used for DHCP within a given network. It takes in three parameters: the network CIDR, the gateway IP, and an optional count indicating the number of addresses to be included in the range. If the count is not provided, it defaults to 20.

2.0.286 Technical description

- **Name**: `generate_dhcp_range_simple()`
- **Description**: This function computes the DHCP range, starting from the IP address after the gateway IP and covering the desired count of IP addresses. If this range exceeds the usable network range, it adjusts the start and end points accordingly within valid limits.
- **Globals**: None
- **Arguments**:
  - `$1`: `network_cidr`: The base network CIDR (e.g. 192.168.50.0/24).
  - `$2`: `gateway_ip`: The gateway IP address (e.g. 192.168.50.1).
  - `$3`: `count`: Optional argument indicating the size of the DHCP range. If this argument is not provided, it defaults to 20.
- **Outputs**: A string containing the start IP, end IP, and lease time (1h) for the DHCP range, separated by commas.
- **Returns**: None directly from the function, but uses `echo` to output information.
- **Example usage**: `generate_dhcp_range_simple "192.168.50.0/24" "192.168.50.1" "50"`

2.0.287 Quality and security recommendations

1. Sanitize inputs: Before processing, ensure that network CIDR block and gateway IP are in the expected formats. This will help prevent potential code injection or data corruption.
2. Error handling: Add logic to handle cases where `ipcalc` or `ip_to_int` functions fail or produce unexpected outputs. This will increase the robustness of the script.
3. Commenting: Inline comments are helpful. Instead considering breaking larger function into smaller functions with descriptive names to enhance readability of the code.
4. Unit testing: Develop suitable unit tests to ensure that the function behaves as expected under a variety of scenarios (both normal and edge cases).
5. Security: Since it doesn't use globals and doesn't modify external state, `generate_dhcp_range_simple` is already quite secure. For added security, consider avoiding the use of `read` and `case` in a subshell spawned by process substitution, which can be susceptible to code injection attacks. A secured alternative can be using a while-loop to process `ipcalc` output one line at a time directly.
6. Validation: Make sure to validate all function inputs as thoroughly as possible to prevent any unauthorized or malicious data from being processed.

2.0.288 `generate_ks`

Contained in `lib/functions.d/kickstart-functions.sh`

Function signature: 7d606fedbb6b82341e778878524f7701844d8ee25178763b26147ed9790a802f

2.0.289 Function overview

The `generate_ks()` function is primarily used in preparing, managing and rendering scripts for host installation. It takes in two arguments, `macid` and `HOST_TYPE`, and exports several environment variables that are crucial in the subsequent steps of the installation process. The function also uses cgi_header_plain and hps_log for logging purposes.

2.0.290 Technical description

- **Name**: generate_ks
- **Description**: The function prepares and executes host installation scripts according to provided arguments.
- **Globals**: [macid: the MAC address and identifier of the host machine. HOST_TYPE: the type of the host OS]
- **Arguments**: [$1: MAC address and identifier of the host machine. $2: The type of the host OS]

- **Outputs**: Logs the state of the host machine and initiates the script for the host installation.
- **Returns**: The function does not return any values, but it does change the state of the host configuration to “INSTALLING”.
- **Example usage**: `generate_ks "macid" "HOST_TYPE"`

2.0.291 Quality and security recommendations

1. Proper validation and sanitization of the input parameters `macid` and `HOST_TYPE` can be done to prevent potential security threats from injection attacks.
2. Add error checking code after exporting each variable and after every invocation of `host_config()` and `cluster_config()`. This can help in identifying issues early on, thereby increasing code robustness.
3. Secure the installation script and related templates, protecting them from unauthorized access or modifications.
4. Replace the TODOs in the script with actual code or remove them if they’re no longer applicable to avoid confusion.
5. Make sure that logging levels and content are correctly set up to prevent leaking of sensitive information.

2.0.292 `get_active_cluster_dir`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: bea86c9b08f47ce60a2602b8a8391990588b8ca0b96c0252ba0070cc3623cfce

2.0.293 Function overview

The `get_active_cluster_dir` function is fundamental to resolving symlinks and retrieving the path of the currently active cluster directory in a system. It defines multiple local variables to find and check the authenticity of this directory, subsequently echoing its path if satisfactory conditions are fulfilled.

2.0.294 Technical description

- **Name**: `get_active_cluster_dir`
- **Description**: This function works to resolve the symlink of the active cluster and retrieve its directory. It does this by storing the symlink and its full path in local variables, verifying their validity, checking if the stored full path is a directory, and if so, outputs the directory. It returns an error if the symlink fails to resolve or the target isn’t a directory.
- **Globals**: None.

- **Arguments**: None.
- **Outputs**: If successful, the function will print the directory of the active cluster. Error messages will be printed to stderr in any of the following cases:
  - The symlink can't be resolved.
  - The active cluster target is not a directory.
- **Returns**: 0 if the function is successful. Returns 1 if the symlink can't be resolved or the target is not a directory.
- **Example Usage**:

```
get_active_cluster_dir
```

2.0.295 Quality and security recommendations

1. Verify if the `get_active_cluster_link` function and the `get_cluster_dir` function used within `get_active_cluster_dir` are secure and optimized. The efficiency of `get_active_cluster_dir` is highly dependent on the performance of these two.
2. Error messages are redirected to stderr, which is a best practice and should be continued.
3. Ensure that this function is running with the right privileges - it should not have more permissions than what is required to read the link and possibly traverse directories.
4. Sanitize the output of the `readlink` and `basename` command to prevent potential path manipulation or symbolic link attacks.
5. Implement unit tests for this function to safeguard it from potential edge cases and bugs.

2.0.296 `get_active_cluster_file`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 6461e5f5814cbc510b1bad68822a5b0d8eb3c58618d50b5418200c62ea6dff01

2.0.297 Function overview

The bash function `get_active_cluster_file()` is designed to retrieve the name of the active cluster saved in a file and output its content.

2.0.298 Technical description

Definition:

- **name**: `get_active_cluster_file`

- **description**: This function retrieves the name of the "active" cluster from the method `get_active_cluster_filename`, assigns it to a local variable `file` and outputs its content, i.e., the active cluster's data. If the `get_active_cluster_filename` method fails, the function will exit and return 1.
- **globals**: None
- **arguments**: None
- **outputs**: The contents of the file retrieved from `get_active_cluster_filename`.
- **returns**: Content of the file if successful, 1 if the `get_active_cluster_filename` fails.
- **example usage**:

```
get_active_cluster_file
```

2.0.299 Quality and security recommendations

1. Include more error handling for situations where file does not exist or it fails to be read by cat command.
2. Ensure that the file reading process is secure and its contents are not accessible to unauthorized users. This can be achieved by setting appropriate permissions on the file.
3. Sanitize output to minimize the potential impact of malicious data.
4. The function should not trust the file's content blindly, it should validate the input before processing it. This is important to prevent possible code injection flaws.
5. Include a more comprehensive documentation, specifying what the function expects as an input and output. This will be beneficial for users of the function. Keep improving the unit tests aiming at improved code coverage.

2.0.300 `get_active_cluster_filename`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 20bf828d972eed107690831358c6e9148058e88842050f23b8944d669bfab00a

2.0.301 Function Overview

The `get_active_cluster_filename` function is designed to find the configuration file for the currently active cluster. It performs the following steps: 1. It obtains the path to the active cluster's directory. 2. It extracts the name of the active cluster from the directory path. 3. It retrieves the cluster configuration file based on the cluster's name. 4. If the file is not found, it prints an error message and returns 1. 5. If the file is found, it prints the file.

2.0.302 Technical Description

- **Function Name**: `get_active_cluster_filename`
- **Description**: This function gets the configuration file name for the currently active cluster.
- **Globals**: None
- **Arguments**: None
- **Outputs**:
  - The path to the configuration file for the active cluster.
  - An error message if the configuration file does not exist.
- **Returns**:
  - Returns `1` when the active directory or the configuration file of the cluster does not exist.
  - Returns `0` otherwise.
- **Example Usage**:

```
filename=$(get_active_cluster_filename)
```

2.0.303 Quality and Security Recommendations

1. To improve readability and maintainability, consider adding comments explaining what each command does.
2. It's good practice to test return values directly in `if` statements rather than relying on `|| return 1` expressions.
3. We should validate user inputs where possible, and handle errors explicitly.
4. Consider using a variable to capture the error message string, which would allow you to change the message in just one place if needed.
5. Make sure to use secure coding practices, such as not making assumptions about input data, checking for null or unexpected values, and handling errors and exceptions as much as possible.

2.0.304 `get_active_cluster_hosts_dir`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 5e8124c7c913768226948ae7120083b2568d70860f21c6599d89e81df548ee7d

2.0.305 Function Overview

The `get_active_cluster_hosts_dir` function in Bash is used to retrieve the hosts directory path for the current active cluster. By appending `/hosts` to the result of the `get_active_cluster_link_path` command, it creates a path to a 'hosts' directory that is expected to be a part of the active cluster's directory structure.

2.0.306 Technical Description

Name: `get_active_cluster_hosts_dir`

Description: This function calls another function `get_active_cluster_link_path` to get the directory path of the active cluster and then appends `/hosts` at the end of the retrieved path. This will provide the 'hosts' directory inside the active cluster directory.

Globals: None

Arguments: No arguments are necessary for this function.

Outputs: The function will output the full path to the 'hosts' directory inside the current active cluster.

Returns: It returns the directory path as a string.

Example Usage:

```
host_dir=$(get_active_cluster_hosts_dir)
echo $host_dir
```

2.0.307 Quality and Security Recommendations

1. Logging: For enhanced debugging and error tracking, consider adding log statements in case of errors when retrieving the active cluster link path.
2. Input Validation: As this function relies on another function, you need to make sure that the `get_active_cluster_link_path` function has appropriate error checks and input validation.
3. Output Validation: It could be beneficial to verify if the returned directory exists and is accessible.
4. Error Handling: Implement error handling to ensure the function behaves predictably in exceptions, including when it cannot correctly fetch the active cluster link path.
5. Documentation: Include comments within the function to describe what the function does. This helps other developers understand the code.
6. Security: Check all file and directory permissions involved in the execution to ensure they're securely configured, as paths and directories often bear security implications in a system.

2.0.308 `get_active_cluster_hosts_dir`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 5e8124c7c913768226948ae7120083b2568d70860f21c6599d89e81df548ee7d

2.0.309 Function Overview

This function is called `get_active_cluster_hosts_dir` and it is used to get the path directory of the hosts from the currently active cluster. It achieves this by calling another function `get_active_cluster_link_path`, concatenating the resulting path with the string '/hosts' and then outputting the final string.

2.0.310 Technical Description

Here's a full description of the various parts of the function:

- **Name**: `get_active_cluster_hosts_dir`
- **Description**: This function generates and outputs the path to the 'hosts' directory of the currently active cluster.
- **Globals**: None.
- **Arguments**: None.
- **Outputs**: The fully formed path to the 'hosts' directory of the currently active cluster.
- **Returns**: None.
- **Example Usage**:

```
path=$(get_active_cluster_hosts_dir)
echo $path # prints the path to the 'hosts' directory of the
 ↪  active cluster
```

2.0.311 Quality and Security Recommendations

1. This function doesn't handle errors and might fail for various reasons (e.g. if `get_active_cluster_link_path` doesn't exist or doesn't output a string). Hence, it would be beneficial to add error handling to this function to make it more robust.
2. Since this function doesn't have any input validations or escaping, it might lead to issues if the output of `get_active_cluster_link_path` were to contain unusual characters (like space or glob characters). A good idea would be to ensure that the paths output by `get_active_cluster_link_path` are sanitized.
3. To ensure better trackability of problems, consider logging errors or warnings whenever the function behaves unexpectedly. This could be done using bash's built-in `echo` or `printf` commands combined with output redirections.

2.0.312 `get_active_cluster_info`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 51d7c68169c79cb4271c76d4914a13afce322dd3dab4f93ccfeba936798070ea

2.0.313 Function Overview

The `get_active_cluster_info` function is a bash function designed to gather and display information about currently active clusters. This function first calls on another function, `_collect_cluster_dirs`, in order to get a list of cluster directories. Then the function checks if there are any active clusters. If there are none, it will print an error message and end the program. If there are active clusters, the function will proceed to print specific information about each cluster.

2.0.314 Technical Description

- **Name**: `get_active_cluster_info`
- **Description**: This function gathers and displays information about currently active clusters.
- **Globals**: `HPS_CLUSTER_CONFIG_BASE_DIR`: It holds the base directory path that the function will check for active clusters.
- **Arguments**: None
- **Outputs**: Error message if no clusters are found, otherwise list of directories stored in `dirs`.
- **Returns**: Returns 1 if no clusters are found, otherwise returns 0.
- **Example Usage**: `get_active_cluster_info`

2.0.315 Quality and Security Recommendations

1. The function should include validation for the global variable `HPS_CLUSTER_CONFIG_BASE_DIR` to ensure it is correctly defined and it points to a valid directory.
2. Use descriptive error messages to allow for easier debugging, and in those error messages include potential solutions for the issues.
3. Ensure that the `_collect_cluster_dirs` function is properly securing and validating the data that it is returning.
4. Check directories for appropriate read permissions before attempt to collect directories.
5. It would be recommended to also provide some form of error handling, for scenarios when the function `_collect_cluster_dirs` isn't defined or fails to execute.
6. Consider adding logging functionality for tracking warnings or errors. This will help in maintaining the system and diagnosing problems.

2.0.316 `get_active_cluster_link`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 8840658f2d249224284adc89b84feddc787683b794880b06d7f3d566dc412130

2.0.317 Function overview

The `get_active_cluster_link` function is a Bash function used to get the active cluster link in a system. It first assigns the output of `get_active_cluster_link_path` function to the `link` variable and then checks if this link is a symbolic link. If not, it prints an error message and returns `1`. If the link is a symbolic link, it simply echoes the `link` - printing the link.

2.0.318 Technical description

- **Name:** get_active_cluster_link
- **Description:** This function is used to retrieve the link to the active cluster in a system. It returns an error if the link does not exist or isn't a symbolic link.
- **Globals:** None
- **Arguments:** None
- **Outputs:** If successful, prints the active cluster link to stdout. If not, an error message is printed to stderr.
- **Returns:** The function returns `0` if the active cluster link is retrieved successfully, and `1` if either no link exists or the link isn't a symbolic link.
- **Example usage:**

  ```
  get_active_cluster_link
  ```

2.0.319 Quality and security recommendations

1. The function does not have any arguments. As such, it would run with any number of arguments provided. To improve quality, error checking can be added to enforce that no arguments are expected.
2. It assumes that `get_active_cluster_link_path` function has already been defined and works correctly. To improve maintainability, there should always be a check if a function exists before it's called.
3. Logging level of error could be standardized. Instead of directly printing to stderr, a logging function can be used to control the logging levels.
4. Error messages printed are currently hardcoded strings. To increase maintainability and readability, these error messages can be converted to constants at the top of the script or inside a configuration file.
5. For security improvements, input validation is a must. For this specific function, checking that the path points to an expected location can prevent symbolic link attacks. For instance, the bash function "realpath" could be used.
6. Lastly, the function needs to handle permissions errors gracefully. It can use defensive programming practices to ensure that the user running the script has the authority to access the link.

2.0.320 get_active_cluster_link_path

Contained in lib/functions.d/cluster-functions.sh

Function signature: 00422606e36c805412e226764ea91ac8d9620442c34f9c3bf83c8de949497756

2.0.321 1. Function Overview

The function get_active_cluster_link_path is used to echo an active cluster link path existing in the ${HPS_CLUSTER_CONFIG_BASE_DIR} directory. It is specifically designed to get the path for the current active cluster configuration file within a High-Performance Computing (HPC) cluster environment. This function does not accept any arguments or modify any global variables.

2.0.322 2. Technical Description

2.0.322.1 Name

get_active_cluster_link_path

2.0.322.2 Description

The get_active_cluster_link_path function is used to output the path of the active cluster config file. It does this by appending the string "active-cluster" to the HPS_CLUSTER_CONFIG_BASE_DIR environment variable using forward slash (/) as the separator to build the file path for the active cluster config file.

2.0.322.3 Globals

[HPS_CLUSTER_CONFIG_BASE_DIR: The directory containing the cluster config files]

2.0.322.4 Arguments

This function does not require any arguments.

2.0.322.5 Outputs

The output is a string indicating the path of the active cluster config file. Example: /path/to/HPS_CLUSTER_CONFIG_BASE_DIR/active-cluster

2.0.322.6 Returns

This function will return 0 on successful execution.

2.0.322.7 Example Usage

```
# Get the path of the active cluster config file
active_cluster_path=$(get_active_cluster_link_path)
echo ${active_cluster_path}
```

2.0.323 3. Quality and Security Recommendations

1. This function does not perform any error checking. Therefore, it's recommended to add error handling to deal with the situation where `HPS_CLUSTER_CONFIG_BASE_DIR` might not be set or could be set to an invalid directory.
2. Confirm that the `active-cluster` file actually exists before attempting to return its path.
3. Protect against Command Injection attacks by sanitizing `HPS_CLUSTER_CONFIG_BASE_DIR` if it's externally controlled data.
4. Always quote your variables - in this case `${HPS_CLUSTER_CONFIG_BASE_DIR}` - to avoid word splitting and globbing. This is important if there are spaces or special characters in the names.
5. Consider adding comments to explain what the function is doing a bit more clearly, especially if other people who are not familiar with the code will be reading or maintaining it.

2.0.324 `get_active_cluster_name`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 56eb8b1d52908fb62f61b716d0ffccedd95bf6c229314f23d9a10385163e0cfd

2.0.325 Function overview

The function `get_active_cluster_name()` is a shell function that retrieves the name of the currently active cluster by utilizing other helper functions. It sets the `dir` variable as the active cluster directory obtained from `get_active_cluster_dir` function. Once the directory is obtained, it retrieves only the last segment from the pathname that denotes the active cluster's name.

2.0.326 Technical description

Name: `get_active_cluster_name()`

Description: This function retrieves the name of the active cluster from the cluster directory's path. It uses the `get_active_cluster_dir` function to get the active directory first, then leverages the `basename` utility to retrieve the active cluster's name.

Globals: None

Arguments: None

Outputs: The active cluster's name.

Returns: It returns 0 on successful execution and 1 if the `get_active_cluster_dir` function fails to execute.

Example Usage:

```
active_cluster=$(get_active_cluster_name)
echo "Active cluster is: $active_cluster"
```

2.0.327 Quality and security recommendations

1. Always quote your variable expansions. For instance, `basename -- "$dir"`.
2. In this function, the `get_active_cluster_dir` function is used, but it cannot handle the case if the function isn't defined. Error handling for this use case should be taken into account.
3. Avoid globals as much as possible as they can produce unpredictable side effects, which can be difficult to debug and maintain.
4. Validate user-defined inputs for security concerns to prevent injection attacks.
5. Write comments for your functions and complex code sections for better readability and maintainability.
6. Always consider edge cases while writing the function. For example, if there is no active cluster, the function should handle this scenario gracefully.
7. Writing unit tests for the functions is a good practice to ensure that they work as expected. It helps to detect function errors, gaps, or missing requirements.

2.0.328 `get_all_block_devices`

Contained in `lib/functions.d/storage_functions.sh`

Function signature: add7deb6a087d72238984be839fb5488e13aff3ae7251f93f2c1814d79162625

2.0.329 Function overview

The `get_all_block_devices` function is used to retrieve all the block devices in a system where their type is 'disk'. It iterates over all block devices in /sys/block directory and which uses a helper function named `get_device_type` to get the type of the device. We get the basename, or the least significant part of the device's path in this instance, to identify the device. If it is a disk, then the device name is printed to the standard output.

2.0.330 Technical description

- **Function Name**: get_all_block_devices

- **Description**: This function retrieves all the block devices whose type is 'disk' from a system.
- **Globals**: devname: contains the name of the device.
- **Arguments**: None.
- **Outputs**: The function outputs to stdout the names of all block devices which are of type 'disk'.
- **Returns**: No explicit return value. Success or failure can be inferred from the lack or presence of output.
- **Example Usage**: `get_all_block_devices`. It needs no arguments.

2.0.331 Quality and security recommendations

1. Validation of the device path: The function directly accesses the /sys/block/ path. The command `basename` can potentially fail if the path does not exist. Hence, validation of the actual device path before processing can improve robustness.
2. Error handling: There is no explicit error handling in case the `get_device_type` returned value is not 'disk' or some other error occurs. It would be beneficial to add error handling mechanisms to improve the robustness of the code.
3. Defensive programming: There are no checks to ensure that the function operates as expected in an abnormal or unanticipated scenario. Therefore, enhancing the function with more checks would increase its resilience.
4. Documentation: There is no comment in the function explaining what it does and how it works. Good documentation makes it easier to maintain and debug the code in the future.

2.0.332 `get_all_hosts_by_keyvalue`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 05e58488251a9243210a5e975395a66c035bc1d2eaf596466d1c914f2e66feda

2.0.333 Function Overview

The function `get_all_hosts_by_keyvalue()` searches for hosts within the active cluster that matches the given key-value pair. It makes use of auxiliary functions and Bash built-in commands to ensure it finds the correct hosts. The hosts are identified by their filenames, which get outputted when found.

2.0.334 Technical Description

- **Name**: `get_all_hosts_by_keyvalue`
- **Description**: This function traverses all host configuration files in the active cluster's directory, searching for the given key-value pairs. It transforms the provided

key to uppercase and the provided value to lowercase before the search to ensure consistent matching.

- **Globals**: None
- **Arguments**:
  - `$1`: search_key - The key to search for in the host configuration files.
  - `$2`: search_value - The value to match with the provided key in the host configuration files.
- **Outputs**: If a match is found, the filename of the host configuration file, sans `.conf` extension, is echoed to stdout.
- **Returns**: 1 if either search_key or search_value is empty, or if the active cluster hosts directory does not exist. Returns 0 if at least one match is found, and 1 if no matches were found.
- **Example Usage**: `get_all_hosts_by_keyvalue "HOST_NAME" "examplehost"`

2.0.335 Quality and Security Recommendations

1. Always ensure the caller is aware that the function is case sensitive and that the case of the input strings is adjusted to fit the data context.
2. To guarantee the security of the cluster, the function should have a way to handle situations when it cannot correctly access or read the host files. It can be set to exit or inform the user to check permissions.
3. To ensure code quality, consider implementing input validation on the `search_key` and `search_value` parameters to ensure they are non-null strings before transforming them. This function currently silently fails if either of the parameters is missing, which could lead to unexpected behaviour.
4. Lastly, handle edge-case scenarios such as incorrectly formatted host files or presence of special characters that might not be considered in the current implementation.

2.0.336 `get_client_mac`

Contained in `lib/functions.d/network-functions.sh`

Function signature: e78a1e2a6af41211f58dd51387958635e90d617e0529987a019dab359db2e31c

2.0.337 Function overview

The `get_client_mac()` function is a bash script function that extracts the MAC address of a client machine on the network. It receives the client's IP address as an argument and initiates an ARP (Address Resolution Protocol) update to ensure the network recognize the existing machine. The script uses regular expressions to extract the MAC address from the returned ARP data and returns the resulting MAC address. It is noted that there is a fallback mechanism to use the 'arp' command if the 'ip

neigh' command does not capture the MAC address. However, this part of the code is commented out in the provided example.

2.0.338 Technical description

Definition block for Pandoc:

- Name: `get_client_mac`
- Description: This function attempts to obtain the MAC address of a client given its IP address.
- Globals: None
- Arguments:
  - $1: IP address of the client machine.
- Outputs: The MAC address corresponding to the given IP address.
- Returns: 1 if the IP address is not provided, 0 otherwise.
- Example usage:

  ```
  get_client_mac 192.168.1.1
  ```

  Output will be the MAC address linked to the IP `192.168.1.1`

2.0.339 Quality and security recommendations

1. Un-comment the block of code that falls back to ARP if the `ip neigh` command does not return a MAC address. This would serve as a backup way of obtaining the MAC address.
2. Include error handling for undesired input such as non-IP formatted strings.
3. Consider including more return codes to specify different kinds of errors to make the function usage and debugging easier.
4. Users should avoid exposing the MAC address retrieved by this function as MAC addresses are unique, and malicious entities can use them to track the network activity of the target machine.
5. It might be safer and recommended to use secure shell protocol (SSH) for any network-commuting commands or utilities.

2.0.340 `get_cluster_conf_file`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: fc7feb39024383e4cb59d1bec6341e1f7248d7dd978aa33bac047e372cc4613e

2.0.341 Function overview

The `get_cluster_conf_file` function takes a cluster name as an argument and returns the path to its configuration file. If no cluster name is provided, it prints out an error message and exits the function.

2.0.342 Technical description

Name:
`get_cluster_conf_file`

Description:
This Bash function takes a cluster name as an argument and returns the path of the configuration file for that cluster. It first checks if a cluster name has been provided, if not an error message is printed and the function exits. It then gets the path to the cluster using the `get_cluster_dir` function.

Globals:
No global variables are used in this function.

Arguments:
- `$1`: The name of the cluster

Outputs:
- If no cluster name is provided, an error message is printed to stderr: `[ERROR] Usage: get_cluster_conf_file <cluster-name>`

- If successful, it prints the path to the cluster's configuration file

Returns:
- `1` if no cluster name is provided or if `get_cluster_dir` function fails - `0` if the function executes successfully

Example usage:
`get_cluster_conf_file my-cluster`

2.0.343 Quality and security recommendations

1. Consider using more descriptive error messages. Instead of `[ERROR] Usage: get_cluster_conf_file <cluster-name>`, something like `[ERROR] Missing required argument: cluster_name` might be more helpful to users.
2. This function trusts that the `get_cluster_dir` function is well-implemented and doesn't validate the return value other than checking for errors. If `get_cluster_dir` could potentially return a harmful or malicious path, then this function will pass it along to the caller.
3. This function assumes that there is always a `cluster.conf` file inside the directory returned by `get_cluster_dir`. Ensure proper error handling if the file does not exist.

4. Avoiding using hardcoded strings (e.g. "/cluster.conf") which could potentially cause problems in the future if there is a change in the configuration filenames or directory structure. Consider using a configuration or properties file to manage these.

2.0.344 `get_cluster_dir`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: f5e9f656e463f433ab0e3bfed0da73d9166cc4e5802421827e55bef022685a0a

2.0.345 Function Overview

The `get_cluster_dir()` function is a simple bash function designed to fetch and echo the directory of a specified cluster. The function takes the name of a cluster as an input, constructs the path to the directory, and outputs the path. If no cluster name is given or if the cluster name is empty, the function outputs an error message and returns with an error status.

2.0.346 Technical Description

- **Name**: `get_cluster_dir`
- **Description**: This function generates the path to a specified cluster directory by concatenating a base directory string with the specified cluster name.
- **Globals**: `HPS_CLUSTER_CONFIG_BASE_DIR`: This global variable is used as the base path to create the full cluster directory path.
- **Arguments**: `$1`: This argument is expected to be the name of a cluster. It is used to construct the full cluster directory path.
- **Outputs**: The function will output the constructed path string to stdout. If no cluster name or an empty string is provided, it will output an error message to stderr.
- **Returns**: Returns 0 if it successfully outputs the full cluster path; returns 1 if no cluster name or an empty string is provided.
- **Example usage**:

```
echo $(get_cluster_dir example_cluster)
# Output: path/to/existing/HPS_CLUSTER_CONFIG_BASE_DIR/example_cluster
```

2.0.347 Quality and Security Recommendations

1. A proper validation of the cluster name should be added before further processing to make sure the input is not malicious and adheres to appropriate naming conventions.

2. To avoid confusion or flawed operations, add checks to ensure that the "HPS_CLUSTER_CONFIG_BASE_DIR" and the constructed directory path actually exist in the file system.
3. To enhance clarity of the function behavior, include a clear and verbose error message to indicate when the function encounters any problem.
4. Be sure to make use of proper permission settings for the directories and files to prevent unauthorized access. This is particularly crucial if this function is part of a larger system that has security implications.
5. Consider defining strings like the error message and base directory as constants at the top of the program or in a configuration file. This enhances maintainability especially when changes need to be made in the future.

2.0.348 `get_cluster_host_hostnames`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 0a454d05c4d3133d2a88b66d07816dd02a6a5ae702c43b33ecd266836eee2f6a

2.0.349 Function Overview

The `get_cluster_host_hostnames` function retrieves the hostnames of the hosts within a specified cluster in a network. The function can optionally filter hosts by the type of host. The function works by reading through each host's configuration file, applying the host type filter if specified, and then outputting the hostname if it exists.

2.0.350 Technical Description

- **Name**: `get_cluster_host_hostnames`
- **Description**: This function gets the hostnames of hosts in a given cluster. It has an optional filter for the host type. The function reads through the configuration files of each host in the cluster, filters out hosts based on the host type filter if specified, and outputs the host's hostname.
- **Globals**: None
- **Arguments**:
  - `$1`: The cluster name. If no value is provided, the function retrieves the active cluster.
  - `$2`: The host type filter. If set, the function only outputs hostnames of hosts that match this type. This input is optional.
- **Outputs**: The function outputs the hostnames of the hosts in the given cluster. It outputs each hostname on a separate line.
- **Returns**: The function returns 0 if it successfully retrieves the hostnames, and 1 if it fails to determine the host's directory.
- **Example Usage**: `bash get_cluster_host_hostnames "my_cluster" "filter"`

2.0.351 Quality and Security Recommendations

1. Implement better error handling. Right now, there's only a single check for if the `hosts_dir` cannot be determined. Additional error checks could be implemented for instance if `list_cluster_hosts "$cluster_name"` fails or returns an empty value.
2. The function reads from host configuration files without performing any validation. Before using the data retrieved from these files, validity checks should be implemented to ensure that the data format matches expectations.
3. The function may echo out an error message directly. It would be beneficial to redirect these error messages to a standard error stream and handle it properly to mimic how a normal UNIX command works.
4. The function uses the `grep -E` command to get certain properties from the host configuration files. While this does work, a more secure option would be to use a more precise tool or command designed for parsing configuration files, such as a standard Linux command or a parser library.
5. Included comments for each block of important code to enhance the readability and maintainability of the function.

2.0.352 `get_cluster_host_ips`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 57b1cadd8f6aaacb69bc449fe7bdb26cd5e773b01b8fcd81a1c730f7042e63c0

2.0.353 Function overview

The `get_cluster_host_ips` function is responsible for retrieving the IP addresses of hosts in a specified cluster. The function takes a single argument, which is the name of the cluster to retrieve host IPs from. If no argument is provided, the function attempts to use the directory of the active cluster. The function works by iterating over the configuration files for each host in the directory of the specified cluster, extracting and outputting the IP address of each host.

2.0.354 Technical description

- **Name**: `get_cluster_host_ips`
- **Description**: This function retrieves the IP addresses for hosts in a specified cluster. If no cluster is specified, it defaults to the active one.
- **Globals**: None explicitly used in the function.
- **Arguments**:
  - `$1: cluster_name:` The name of the cluster from which to retrieve the list of hosts and their IP addresses.
- **Outputs**: List of IP addresses of all hosts in the specified or active cluster.

- **Returns**:
  - `1`: If it cannot determine the hosts directory.
  - `0`: Successfully retrieves the IP addresses.
- **Example usage**:

```
$ get_cluster_host_ips my_cluster
192.168.1.1
192.168.1.2
```

2.0.355 Quality and security recommendations

1. Perform validation on the argument: The function should verify whether the provided argument is a valid cluster name or not.
2. Error handling enhancements: The function could also benefit from more robust error handling, for example by making sure that the `grep`, `cut`, and `tr` commands are successful.
3. Security enhancements: Be aware that if an attacker can manipulate the content of host's configuration files, they may be able to inject malicious data. Ensure only authorized personnel can modify these files.
4. Additional return codes: Implement more specific return codes that can indicate various failure points within the function for easier troubleshooting.
5. Documentation: Comment on any global variables and their usage within the function for clarity.

2.0.356 `get_device_bus_type`

Contained in `lib/functions.d/storage_functions.sh`

Function signature: 71e77c52eb2ba1d481c9ef51c928878b08e2c1088eeb0bf2fbe433c62633a476

2.0.357 Function overview

The function `get_device_bus_type()` takes one argument, which represents a device, and returns the device bus type based on the device name or device property defined in the local environment. If the device name starts with `/dev/nvme`, it will echo "NVMe". If not, it then checks if the device is rotational, if it's not rotational, it echoes "SSD", otherwise, it echoes "HDD".

2.0.358 Technical description

- **Name:** `get_device_bus_type`
- **Description:** This function identifies the type of bus a provided device is using. It checks whether the input is a Non-Volatile Memory Express (NVMe) device, a Solid State Drive (SSD), or Hard Disk Drive (HDD) by analyzing its name or its rotational property.

- **Globals:** None
- **Arguments:**
  - `$1:` `dev` A string that represents the device.
- **Outputs:**
  - If the device name starts with `/dev/nvme`, it outputs “NVMe”.
  - Otherwise, if the device is not rotational, it outputs “SSD”.
  - Otherwise, it outputs “HDD”.
- **Returns:** None
- **Example usage:** `get_device_bus_type /dev/nvme0n1`

2.0.359 Quality and security recommendations

1. Always validate the input to ensure that the device provided exists in the system.
2. Consult the device properties from a trusted source, or directly from the system if possible, instead of purely relying on the device name pattern.
3. Maintain the single-responsibility principle. The function may benefit from being split into multiple smaller functions, each with its own responsibilities: one for checking if the device is NVMe, another for checking if the device is an SSD, and another for checking if the device is an HDD.
4. Always handle potential errors or exceptional cases to avoid unexpected behaviors. In this function, an else-case would be beneficial to handle situations where the device is neither an NVMe device, SSD, nor HDD.

2.0.360 `get_device_model`

Contained in `lib/functions.d/storage_functions.sh`

Function signature: ec59456e4d8546a811f0f4533daf51da08474fb4ada4f1efb264e30d2fea7091

2.0.361 Function overview

The `get_device_model()` function in Bash is used to get the model of a specific device. It takes a device identifier as an argument and then accesses the corresponding system information to return the model of the device. If the function cannot find the specified device, or if any other error occurs, it will return a string saying “unknown”.

2.0.362 Technical description

- **Name:** get_device_model
- **Description:** This function retrieves the model of a device given its identifier. It looks in the “/sys/block” directory, removes any whitespace, and then returns the

model of the device. If the system cannot find the model for any reason, it reports "unknown".

- **Globals:** None.
- **Arguments:**
  - `$1: dev` - This is the identifier of the device whose model is being retrieved
- **Outputs:** Model of the device or "unknown" if the device model can't be found.
- **Returns:** The function returns the model of the device or "unknown" if there is an error or if the device can't be found.
- **Example usage:**

```
model=$(get_device_model sda)
echo $model
```

2.0.363 Quality and security recommendations

1. Implement error checking for the `cat` command.
2. Check the validity of the input device identifier before processing.
3. Provide a more descriptive error message.
4. Sanitize the input to avoid command injection vulnerabilities.
5. Handle device names that contain spaces correctly.
6. Implement proper logging to understand the behavior of the function in case of failures.

2.0.364 `get_device_rotational`

Contained in `lib/functions.d/storage_functions.sh`

Function signature: 1002aec4163a82a48171eea735ad8700633333695a4b75dcb9ecc3317d14222f

2.0.365 Function overview

The `get_device_rotational()` function in Bash is designed to fetch the rotational status of the specified device. This status is derived from the `sysfs` filesystem and signals whether the device uses rotational storage media (like a traditional hard drive) or not (like an SSD). If the function is unable to retrieve this information, it defaults to returning `"1"`.

2.0.366 Technical description

- **Name**: `get_device_rotational()`
- **Description**: This function fetches and prints the rotational status of a specific block device. '1' implies the device uses rotational media, while '0' indicates use of non-rotational media.
- **Globals**: None.

- **Arguments**: `$1: dev` — The device whose rotational status is to be fetched.
- **Outputs**: The rotational status ('1' or '0') of the specified device.
- **Returns**: Nothing.
- **Example Usage**: `get_device_rotational sda`

  In the example above, 'sda' is the block device for which the rotational status is desired.

2.0.367 Quality and security recommendations

1. Consider using more descriptive variable names to improve code readability.
2. Remember to properly quote your variables to avoid word-splitting or pathname expansion.
3. Always use `#!/bin/bash` or `#!/usr/bin/env bash` for writing bash scripts, not `#!/bin/sh`, because it may not actually link to `bash` on many systems, leading to unexpected behavior.
4. You should check if the device path exists in the sys filesystem as a preliminary step before attempting to fetch the rotational status.
5. Consider error handling for cases where the provided device name does not exist.

2.0.368 `get_device_serial`

Contained in `lib/functions.d/storage_functions.sh`

Function signature: 8e1b210627973a0cc629e646a16460819b61ae08954c5dba162358f2b6cb4532

2.0.369 Function overview

The function `get_device_serial` is used for extracting the serial number from a specific device in a Linux system. This is accomplished by querying the Udev device manager using the `udevadm info` command with the particular device and parsing the output to get the serial number.

2.0.370 Technical description

- **name**: `get_device_serial`
- **description**: This function takes in a device (a local variable `dev`) as an argument, runs a query for its properties using `udevadm`, and extracts its ID_SERIAL value (the device's serial number). If no serial is found, it returns "unknown".
- **globals**: No global variables are used in this function.
- **arguments**:
  - `$1: dev`, the name of the device to get the serial number from.
- **outputs**: Either the device's serial number or the string "unknown" if no serial number is found.

- **returns**: The function doesn't have a `return` statement, its output is a side-effect of the function.
- **example usage**: `get_device_serial /dev/sda`

2.0.371 Quality and security recommendations

1. Add error checking and handling for non-existent devices or permission issues when running the `udevadm` command.
2. Currently, the function silently defaults to "unknown" when the device serial cannot be retrieved. This could be enhanced by incorporating a verbose mode or a warning message to inform the user about possible issues.
3. To avoid potential command injection, ensure that the provided input is properly validated and sanitized before it is used.
4. Prefer using the `printf` function over `echo` for compatibility and predictability reasons.
5. Always use double quotes around variable substitutions to avoid word splitting and pathname expansion. For instance, `--name="$dev"` instead of `--name=$dev`. This is already correctly done in the provided function.

2.0.372 `get_device_size`

Contained in `lib/functions.d/storage_functions.sh`

Function signature: 1d4e207d13b80b5424975cd10a3ed7197f78e6832fbbf6662e417abdd6def6d2

2.0.373 Function Overview

The `get_device_size` function is a bash function that accepts a device name as an argument and then obtains the size of the device using the `lsblk` command. If the device size cannot be determined, it will output "unknown".

2.0.374 Technical Description

- **Name**: `get_device_size`
- **Description**: This function retrieves the size of a specified device. It utilizes the `lsblk` command and, in the event the device size cannot be determined, it outputs the string "unknown".
- **Globals**: None
- **Arguments**:
  - `$1`: The device whose size is to be determined. It is usually a block device-like "/dev/sda".
- **Outputs**: The size of the device. If the size can't be determined, it outputs the string "unknown".

- **Returns**: The status of execution of the last command executed, typically the `lsblk` command.
- **Example Usage**: `get_device_size "/dev/sda"`

2.0.375 Quality and Security Recommendations

1. Since the function performs operations on block devices, it should have proper error checking and handling. This would help in maintaining the integrity of the device and prevent any data loss.
2. Validate the input to the function to ensure that it is a valid device name.
3. The function echoes "unknown" when it can't determine the device size. It might be better to return a specific error code for this situation.
4. Consider adding checks to ensure that the required utilities (`lsblk` in this case) are available on the system. This could prevent errors from occurring due to missing utilities.
5. It's always a good idea to run scripts as a non-root user when possible. If this function needs root privileges, make sure you handle that securely.

2.0.376 `get_device_speed`

Contained in `lib/functions.d/storage_functions.sh`

Function signature: a3925bc33db6a5252197ab449a78edcdfb4d9117f6e067cedb4b22c2eaea3a3e

2.0.377 Function overview

The function `get_device_speed` measures the read speed of a particular device.

2.0.378 Technical description

- **name:** `get_device_speed`
- **description:** This bash function measures and outputs the read speed of a given device. It reads data from the specified device using the `dd` command and pipes the output to the `grep` command to filter the speed data. In case the speed data can't be retrieved, it prints "N/A".
- **globals:** None
- **arguments:**
  - `$1`: This refers to the device for which the read speed is being measured.
- **outputs:** It outputs the read speed of the provided device in the format '[0-9.]+ MB/s', else if it can't fetch the data, it prints "N/A".
- **returns:** It won't return any standard exit codes, but the output of the speed of the device.
- **example usage:**

```
get_device_speed "/dev/sda1"
```

2.0.379 Quality and security recommendations

1. Add proper error handling: We should have a way to handle situations where the device does not exist, is not readable or the dd command is not available.
2. Return error codes: This function doesn't have a return value, it just emits the output. To improve its usability in scripts, it would be better to return distinct codes for different error conditions.
3. Check for needed utilities: It would be great if function checks for presence of dd and `grep` commands at the start of execution.
4. Input sanitation: Inputs should be sanitized or validated to prevent potential security risks. Validate the device name accordingly.
5. Improve the output: To increase the usability of the function, it could return both the raw speed data and a human-readable string.

2.0.380 `get_device_type`

Contained in `lib/functions.d/storage_functions.sh`

Function signature: c04dccee20f8398f1cc00c1f339464f78c0ea09416025a9627dcbd45477ea68d

2.0.381 Function Overview

This function, `get_device_type()`, utilizes the `udevadm` command to fetch specific device properties. Given a device ID as a parameter, it queries for properties and filters out the device type. By design, if no correct device ID is given or if the `udevadm` fails, the function defaults to returning "disk".

2.0.382 Technical Description

- Name: `get_device_type`
- Description: Fetches and returns the type of a device by utilizing the device's unique ID. If a type is not found or the function fails, it defaults to returning "disk".
- Globals: None
- Arguments: [$1: The unique ID of the device for which the type is to be found]
- Outputs: The type of the given device. Prints "disk" if no correct ID is given or if the function fails.
- Returns: 0 on successful execution, non-zero on error.
- Example Usage:

```
device_type=$(get_device_type /dev/sda1)
echo $device_type
```

2.0.383 Quality and Security Recommendations

1. It is recommended to add formal error handling to help diagnose potential issues or incorrect inputs more easily. Relying solely on a default return value can mask

actual errors.
2. The usage of `grep` and `cut` to parse the output of `udevadm` assumes a specific output format and might break if the output or format changes in the future. A more robust parser could be considered.
3. All inputs, even if they are supposed to be device IDs, should be sanitized to prevent potential Bash command injection issues.
4. A detailed comment describing the function, its parameters, and its return values can improve maintainability and readability of the code.

2.0.384 `get_device_usage`

Contained in `lib/functions.d/storage_functions.sh`

Function signature: 859cf1265955125053e13b61c6c4bfa14f3715f652ae3ee914eb7e3927870233

2.0.385 Function overview

The function `get_device_usage()` is used to fetch the usage details about a device. The function accepts a device identifier as an argument and returns a comma-separated string of mount points where the device is in use. If the device is not used anywhere, it returns "unused".

2.0.386 Technical description

- **Name**: get_device_usage
- **Description**: This function utilizes Linux command line utilities to decipher the usage of a given device. The device identifier is passed as an argument and usage details are then obtained with the 'lsblk' command. The list of places where the device is in use is compiled into a comma-separated string. If the device is not currently in use anywhere, "unused" is returned.
- **Globals**: None
- **Arguments**:
  - $1: Device identifier (e.g., /dev/sda1)
- **Outputs**: Comma-separated string of device usage locations or "unused" if the device has no usage records.
- **Returns**: 0 on success.
- **Example Usage**:

```
usage=$(get_device_usage "/dev/sda1")
echo $usage
```

2.0.387 Quality and security recommendations

1. Input validation: This function currently performs no validation on input. It would be beneficial to add checks to ensure that the argument passed is actually a valid

device identifier.
2. Error handling: Updates could be done to the function to make it handle, recover from, or report any errors that may occur during the execution of command line utilities used.
3. Use of unassigned variables: In the current format, if `lsblk` fails to execute, `usage` will be unset causing an 'unbound variable' error to be thrown. Consider setting a default value for `usage` to prevent this.
4. Secure handling of command substitution: Use the "$()" construct for command substitution to avoid potential security issues with backticks. The current implementation already follows this recommendation.

2.0.388 `get_device_vendor`

Contained in `lib/functions.d/storage_functions.sh`

Function signature: e635f4842a790d321a2d2bf3371ad26f62f2881ced6effdff7eaea8887f48cf1

2.0.389 Function overview

The `get_device_vendor` function aims to find out and return the vendor information of a given device specified by the argument passed to the function. It is a bash function implemented to dive deep into system files to extract this information. If the information is not available or if the system encounters any error while trying to find out the information, it simply returns an "unknown" string.

2.0.390 Technical description

- **Name:** get_device_vendor
- **Description:** This function retrieves the vendor information of a given device in a Linux-based operating system.
- **Globals:** None.
- **Arguments:** [`$1`: The device (e.g., `/dev/sda`) for which to retrieve the vendor information.]
- **Outputs:** Prints vendor information to the standard output. If the vendor information could not be obtained, prints "unknown".
- **Returns:** None.
- **Example usage:** `get_device_vendor /dev/sda` *This will return the vendor information of the sda disk.*

2.0.391 Quality and security recommendations

1. Validate the input argument as a valid device before proceeding with the command to prevent unexpected behavior or potential security breaches.

2. Adding error checks and handling could improve the function further. Right now, if anything goes wrong, the function will simply print “unknown” which might not be the most informative way to handle errors in some cases.
3. Check for the appropriate permissions before trying to access system files. Your script might not work without the right permissions or in a restricted environment.
4. Include more detailed comments in the code to explain decision reasons and to clarify hard-to-understand parts. Even though the code looks simple, good commenting is a strong marker of software quality.
5. You should consider setting a stricter error handling policy, like `set -euo pipefail`, to make the script exit on the first error it encounters.

2.0.392 `get_distro_base_path`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: b61e67918c748d5677a48cc563c5168c106e06af5e5343a2daab1bb78d3ec7aa

2.0.393 Function Overview

The `get_distro_base_path()` function in Bash is used to retrieve the base path of a given operating system distribution. It achieves this by accepting the operating system ID and the type of the path as parameters. If the operating system ID does not have a configured repo path or is not specified, the function logs an error and returns a failure code. The repo path is then determined based on the path type which can be either `mount`, `http`, `relative`, or others. If the path type is not recognized, it logs an error and returns a failure.

2.0.394 Technical Description

- **name**: get_distro_base_path
- **description**: Function to retrieve the base path of a specified operating system distribution.
- **globals**:
  - `repo_path`: The path to the repository.
- **arguments**:
  - `$1: os_id`: The identifier of the operating system.
  - `$2: path_type`: The type of the path. If not specified, it defaults to `mount`.
- **outputs**: Depending on the `path_type`, it returns
  - The mount path of the repo.
  - The http path of the repo.
  - The relative path of the repo.
- **returns**: Returns 1 if the os_id is not specified or does not have a repo_path set or if the path type is unknown.
- **example usage**:

```
get_distro_base_path ubuntu mount
```

2.0.395 Quality and Security Recommendations

1. Avoid using global variables as much as possible because they cause side effects which are hard to track. Consider using function arguments instead.
2. Always validate function arguments before use. If an argument is undefined or in an incorrect format it can cause bugs which are hard to debug.
3. Implement more error handling. If the os_config function fails, your script will continue even though there's likely a problem.
4. Use more descriptive error messages. Instead of "O/S $os_id does not have a repo_path set", consider a message like "Unable to find a repository path for the operating system with the ID: $os_id".
5. Add input sanitation for inputs to protect against Command Injection, a common security vulnerability.

2.0.396 `_get_distro_dir`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: 9eb292c5d01c6e6adc45155806a06d1e4ac6df2ccb7a483363a9daf07aea7a49

2.0.397 Function Overview

This Bash function, `_get_distro_dir()`, is designed to fetch and echo the value of the variable `HPS_DISTROS_DIR`. The key role of this function is to provide a way to retrieve the directory path of the distribution files.

2.0.398 Technical Description

```
- Name: _get_distro_dir
- Description: This function is used to echo the value of the
 ↪  variable `HPS_DISTROS_DIR`, mainly serving to fetch the
 ↪  directory path of distribution files.
- Globals: [ HPS_DISTROS_DIR: This is a global variable that
 ↪  stores the path to the directory of distribution files. ]
- Arguments: [ No arguments expected ]
- Outputs: The function outputs the path of the distribution
 ↪  directory stored in the `HPS_DISTROS_DIR` variable.
- Returns: It doesn't return values except for the stdout.
- Example usage: `_get_distro_dir`
```

2.0.399 Quality and Security Recommendations

1. Make sure the `HPS_DISTROS_DIR` variable is always assigned a valid directory path value.

2. Validate and sanitize directory paths wherever possible to prevent potential mishandling.
3. It is advisable to avoid using uppercase for function names to prevent any potential conflicts with shell variables as by convention, environment variables (PAGER, EDITOR, SHELL) and internal shell variables (BASH_VERSION, IFS) are uppercase. It's suggested to use lowercase for function names.
4. Error handling could be added to this function. For instance, one could check if HPS_DISTROS_DIR actually exists and is accessible before echoing it. If the check fails, one can return an error code and log a message explaining what went wrong.
5. While not a security concern, commenting and documenting the function would provide a better understanding of it to other developers or even the future you.

2.0.400 _get_distro_dir

Contained in lib/functions.d/os-function-helpers.sh

Function signature: 9eb292c5d01c6e6adc45155806a06d1e4ac6df2ccb7a483363a9daf07aea7a49

2.0.401 Function Overview

This function, _get_distro_dir, is a simple Bash function that prints out the value of a global variable named HPS_DISTROS_DIR. This global variable is expected to store the directory path of the distribution files in a system.

2.0.402 Technical Description

- **name**: _get_distro_dir
- **description**: A Bash function that echoes the value of a global variable HPS_DISTROS_DIR.
- **globals**: [HPS_DISTROS_DIR: Directory path where distribution files are located.]
- **arguments**: None
- **outputs**: Prints the value of HPS_DISTROS_DIR to the standard output.
- **returns**: None
- **example usage**:

  ```
  _get_distro_dir
  ```

2.0.403 Quality and Security Recommendations

1. Ensure that HPS_DISTROS_DIR is set: The function assumes that HPS_DISTROS_DIR has already been initialized. It is important to check

whether this global variable exists and if not, handle its absence in some way.
2. Check for proper directory path: If a directory path is supposed to have a particular format, checks could be added to validate the value of HPS_DISTROS_DIR.
3. Control access rights: As this function exposes the value of a global variable, the accessibility of this function should be controlled to prevent unintended information disclosure.
4. Sanitize output: Before directly using the output from the function, consider sanitizing or checking it, especially if it will be used as part of a file path or a shell command. This can help prevent path traversal or code execution vulnerabilities.

2.0.404 `get_distro_url`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: d5e5cb266493aa86db394cb7cbb071c14d0e30e84fcc69d8ddf988f413d59f36

2.0.405 Function overview

The Bash function `get_distro_url` constructs a complete URL to access distribution files on a specific server. It requires an operating system identifier as a parameter and optionally, a server address. If the server address is not provided, it will use a global server address (SERVER_ADDR).

2.0.406 Technical description

- **Name:** `get_distro_url`
- **Description:** Constructs a fully qualified URL for accessing distribution files of a specific OS on a particular server. This function makes use of another function `get_distro_base_path` to get the base path of the distribution files.
- **Globals:** `SERVER_ADDR: The server address to be used if not provided as a function argument.`
- **Arguments:**
  - `$1: The OS ID for which the distribution files URL is required.`
  - `$2: The server address to access the distribution files. If not provided, the global var SERVER_ADDR will be used.`
- **Outputs:** Prints the complete URL to stdout.
- **Returns:** None. In case of an error (like the absence of necessary globals or arguments), the function may not work as expected or may throw an error message.
- **Example usage:**

```
# Get the URL with provided server address
get_distro_url "ubuntu" "example.com"
```

```
# Get the URL with the global SERVER_ADDR
get_distro_url "debian"
```

2.0.407 Quality and security recommendations

1. Implement error checking to make sure all required variables and arguments are provided. If not, return an appropriate error message or code.
2. Validate provided arguments, especially if they are user-supplied (to avoid issues like code injection).
3. Ensure the invoked `get_distro_base_path` function is well-tested and secure.
4. Use HTTPS instead of HTTP where possible for improved security.
5. Always quote your variables to prevent word splitting and pathname expansion.

2.0.408 `get_external_package_to_repo`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: a00cf7324f6795dbbcefc882d9a0346242d31cdeaef0ed00e85c79720244ebc9

2.0.409 Function overview

The `get_external_package_to_repo` function is designed to download a package from a specified URL and save it to a specified repository path. It performs various checks to ensure that the package URL and repository path are valid, that the repository directory exists, and that the package file has a .rpm extension. It logs various information and error messages during its operation, and it uses the `curl` command to download the package.

2.0.410 Technical description

- **Name**: `get_external_package_to_repo`
- **Description**: This function downloads an RPM package from a specified URL to a specified repository path. It checks for correct usage and the existence of the target directory, and it verifies that the URL points to an RPM file. It logs information about its operation and any errors it encounters.
- **Globals**: None.
- **Arguments**:
  - `$1`: URL from which to download the RPM package.
  - `$2`: Path to a directory that will hold downloaded package.
- **Outputs**: Logs information and error messages to standard output.
- **Returns**:
  - `0` if the package is successfully downloaded
  - `1` if incorrect usage

  - 2 if target directory does not exist
  - 3 if URL does not point to an RPM file
  - 4 if it fails to download package URL.
- **Example usage**: `get_external_package_to_repo "http://example.com/package.rpm" "/path/to/repo"`

2.0.411 Quality and security recommendations

1. Use absolute paths for the repository path to avoid potential confusions or errors with relative paths.
2. Validate the URL more thoroughly. Currently, it only checks if the URL ends with `.rpm`. More comprehensive validation would be beneficial.
3. Use the `-s` (silent) option with `curl` to suppress unnecessary output and only display important information.
4. Consider using a more secure protocol, such as HTTPS, for downloading the package to ensure the integrity and security of the download.
5. Add more detailed logging, including timestamps and the full file path of the downloaded files, to allow easier troubleshooting.
6. Rather than relying on return codes, consider propagating more detailed error information to give invokers more context of failures.

2.0.412 `get_host_conf_filename`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 8b543e00f7c0c713c4d8c537217c6a9c6addf01207ee1dfb10ecf00e0041e3e7

2.0.413 Function overview

The `get_host_conf_filename` function accepts a MAC address as an argument and proceeds to look for a corresponding configuration file in the active cluster hosts directory. If the MAC address is not provided, the function logs an error and returns. If the active cluster hosts directory cannot be determined, the function logs an error and returns. The function also logs errors and returns in the case that the configuration file does not exist or is not readable. When successful, the function outputs the path of the configuration file and returns zero.

2.0.414 Technical description

- **Name:** `get_host_conf_filename`
- **Description:** This function is used to retrieve the path of a configuration file for a given MAC address in the active cluster hosts directory.
- **Globals:** None

- **Arguments:**
  - `$1: MAC address` - This is expected to be a MAC address as a string.
- **Outputs:** If successful, this function will output the path to the configuration file.
- **Returns:**
  - `0` if the function successfully retrieves the path of the configuration file.
  - `1` if either the MAC address is not provided or the active cluster hosts directory cannot be determined.
  - `2` if the configuration file does not exist or is not readable.
- **Example usage:**

  ```
  get_host_conf_filename "00:0A:95:9D:68:16"
  ```

2.0.415 Quality and security recommendations

1. Validate the format of the MAC address not only for presence but also for the correct syntax.
2. Handle exceptions for the `get_active_cluster_hosts_dir` function call.
3. Consider testing if the configuration file is not just readable but also in a valid format.
4. In addition to logging, consider more user-friendly error handling that informs what actions the user should take.
5. Securing the directory and files that the function accesses to prevent unauthorized changes.
6. Consider using more general exit status codes to increase portability across different systems.

2.0.416 `get_host_mac_by_keyvalue`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 4e6ce4d36c6772f4c729ca17b2e92f175a648b408ef7b40a3be7b78ad7931ede

2.0.417 Function overview

The `get_host_mac_by_keyvalue` function is used to find and echo the MAC address of a host based on a provided key-value pair. If the key-value pair is found in any of the host configuration files in the active cluster hosts directory, the function will echo the name of the file (which is the MAC address of the host) and return with success.

The function performs a case-insensitive search and will also clean up any quotes present in the value. If either the search key or value is not provided, the function will return an error.

2.0.418 Technical description

Definition block for `get_host_mac_by_keyvalue`:

- **Name**: `get_host_mac_by_keyvalue`
- **Description**: Searches through the configuration files in the active cluster `hosts` directory for a specified key-value pair. If found, the function will echo the filename (which corresponds to the MAC address of that host).
- **Globals**:
  - `VAR: desc`: None
- **Arguments**:
  - `$1: search_key`: The key to search for. It is converted to uppercase for a case-insensitive match.
  - `$2: search_value`: The value to be matched with the search key. It is converted to lowercase for a case-insensitive match.
- **Outputs**:
  - If a match is found, the function echoes the filename (which corresponds to the MAC address of the host).
- **Returns**:
  - `0`: Success, if a match is found.
  - `1`: Error, if a match is not found or any of the search parameters are missing.
- **Example usage**:

```
get_host_mac_by_keyvalue "HOSTNAME" "myhost"
```

2.0.419 Quality and security recommendations

1. Instead of executing the function in the current shell environment, consider using a subshell to encapsulate the variables and limit their scope, enhancing security.
2. Implement logging to record function usage and errors for easier debugging and tracking of potential security issues.
3. Validate the inputs to prevent injection attacks or to handle special characters properly.
4. Add more error checks and provide clear error messages for easier debugging and better user experience.
5. Consider handling more edge cases, such as when there are multiple matches found.

2.0.420 `get_host_os_id`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 0552306e684ccb3b1ac87108c1a0b366ce229ba2ff30177629fecb3c1e56c566

2.0.421 Function overview

The `get_host_os_id()` function is used to determine and return the operating system identification related to a specified host based on its MAC address. It first retrieves the host type and architecture. It ensures that we have a host type; if not, it returns an error message. If no architecture is specified, the function defaults to `x86_64`. It then retrieves the corresponding OS ID from cluster configuration. If the OS ID is not found, it falls back to a non-architecture-specific configuration. If valid OS ID is not found or does not exist in the registry, it returns an error message; otherwise, it outputs the OS ID.

2.0.422 Technical description

- **Name:** get_host_os_id
- **Description:** This function obtains the operating system ID for a host using its MAC address.
- **Globals:** None
- **Arguments:**
  - `$1: mac`, MAC address of a host.
- **Outputs:** Prints the OS ID if found.
- **Returns:** Returns 0 on successful retrieval of OS ID, 1 if OS ID retrieval fails, or if the required host type or OS configuration was not found.
- **Example usage:**

```
# Assume 00:0a:95:9d:68:16 is MAC address of a host
get_host_os_id "00:0a:95:9d:68:16"
```

2.0.423 Quality and security recommendations

1. Validate MAC address input: There should be a check for the validity of the MAC address passed to the function.
2. Error handling: Consider creating a dedicated function for error handling that centralizes all error messages and makes the code easier to read and maintain.
3. Code documentation: Include inline comments for complex operations to make the function more understandable.
4. Securing output: In the echo command that displays `$os_id`, ensure that the output is treated properly to prevent any potential command injection.
5. Handling unexpected output: Add checks for empty or unexpected responses from `host_config` and `cluster_config` to increase the robustness of the function.

2.0.424 `get_host_os_version`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 56ca07cdef820435f1b11fd42dcbf53229c722f8fa266f2cee5b4b33679d6167

2.0.425 Function overview

The `get_host_os_version` function in Bash is utilized as a method to determine the version of the operating system on a specified host machine. It does this by first using the `get_host_os_id` function to pull the OS ID based on the machine's MAC address, then the `os_config` function to retrieve the version for that OS ID.

2.0.426 Technical description

- **Name:** `get_host_os_version`
- **Description:** This function utilizes the MAC address of a target host machine to retrieve its OS ID and, subsequently, its OS version. Specifically, it uses the `get_host_os_id` function to get the OS ID from a given MAC address and passes this ID to `os_config` function together with the "get" and "version" arguments.
- **Globals:** None used in this function.
- **Arguments:**
  - `$1: Host machine's MAC address. The MAC address identifies a specific machine in the network.`
  - `$2: Not used in this function.`
- **Outputs:** If successful, it will output the OS version of the host. Action is undefined if the OS ID cannot be retrieved from the MAC address.
- **Returns:** Returns `1` if there is any error (i.e., if no OS ID is returned), else retrieves and returns the OS version of the host.
- **Example Usage:**

  ```
  $ get_host_os_version 00:00:00:00:00:00
  ```

2.0.427 Quality and security recommendations

1. This function relies heavily on the `get_host_os_id` function, so make sure that function is thoroughly tested and fails gracefully if anything goes wrong.
2. Consider adding error checking for invalid or improperly formatted MAC addresses.
3. Be aware that MAC addresses can sometimes be spoofed, which could lead to misleading results. Additional security measures may be needed to confirm the identity of the host machine.
4. If the `os_config` function is capable of modifying system settings, ensure it is properly secured against unauthorized use.
5. The returned OS version should be sanitized if it's going to be used as an argument for other commands, to protect against command injection attacks.

2.0.428 `get_host_type_param`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 2aa086f62cbb99876d8c7312d176dfce1f88af4d006d3f9a64537fa96e3f9008

2.0.429 Function overview

The `get_host_type_param()` function in Bash is designed to retrieve a specific value from an associative array, based on the provided key. This function takes in two parameters: the name of the associative array, and the key for which the value should be retrieved. The value corresponding to the given key is then echoed out, allowing the function's output to be captured and used elsewhere in the script.

2.0.430 Technical description

- **Name:** `get_host_type_param()`
- **Description:** This Bash function retrieves a specific value from an associative array. The name of the associative array and the key are provided as input arguments. The function uses Bash's variable reference (`declare -n`) to reference the associative array, and then uses array indexing (`${ref[$key]}`) to fetch the value corresponding to the provided key.
- **Globals:** None
- **Arguments:**
  - `$1 (type)`: The name of the associative array from which to retrieve the value.
  - `$2 (key)`: The key for which to retrieve the value from the associative array.
- **Outputs:** Echos out the value from the associative array that corresponds to the provided key.
- **Returns:** Does not return value.
- **Example usage:** `get_host_type_param "server_param" "ip_address"`

2.0.431 Quality and security recommendations

1. It's important to ensure the arguments passed into this function (the name of the associative array and the key) are not controlled by untrusted user input, as this could potentially lead to unintended behavior.
2. Consider adding error checks in the function to handle cases where the provided associative array or key might not exist. Currently, if either the array or the key doesn't exist, the function won't return an error or warning, which might lead to bug diagnosis challenges.

3. It might be beneficial to enclose all variable references, including array indices, within double quotes. This will prevent word splitting and pathname expansion, which could lead to unexpected results in some cases.

2.0.432 `get_interface_network_info`

Contained in `lib/functions.d/network-functions.sh`

Function signature: 4e173360adff11eb87d4f357c81a5a4827f74ea723adff69c69fd4d1c4dcae15

2.0.433 Function overview

The `get_interface_network_info` function is designed to gather network interface information on a Linux system. The function accepts a network interface as an argument and retrieves several essential parameters such as interface's IP Address, CIDR notation, IP/CIDR combination, network address. It then outputs this information in a formatted string. The function uses external utility `ipcalc` to calculate network subnet from the IP/CIDR combination. If the network interface does not have an IPv4 address or the `ipcalc` tool is not installed or the network subnet couldn't be calculated, the function logs an error message via `hps_log` function and returns `1`.

2.0.434 Technical description

- **Name:** `get_interface_network_info`
- **Description:** This function retrieves important networking information related to a specified network interface and outputs it in the following format: `interface|ipaddress|cidr|ip_cidr|network`
- **Globals:** None
- **Arguments:**
  - `$1`: Name of a network interface
- **Outputs:** A string formatted as `interface|ipaddress|cidr|ip_cidr|network`
- **Returns:** `0` on successful execution; `1` when the network interface does not have an IPv4 address, `ipcalc` utility is not installed, or calculation of network subnet fails.
- **Example usage:** `get_interface_network_info eth0` This will output the eth0 interface details in the specified format if successful or log an error message and return 1 if unsuccessful.

2.0.435 Quality and Security Recommendations

1. Error Handling: Continue to use explicit error messages to ensure that failures due to misconfigurations, missing dependencies, or out-of-resource conditions are understood and addressed promptly.

2. Dependence on External Utilities: Currently, this function depends on `ipcalc` utility. If it's absent, the function fails. To improve this, consider including a fallback mechanism when `ipcalc` is not available or implement its functionality within the script avoiding the dependency altogether.
3. Validation of Input: Add robust validation for the input parameter. Ensure the network interface exists on the system before proceeding with the function.
4. Code Comments: Maintain good commenting practice throughout the code for better readability and maintainability. Comments should explain why something is done, not what is done.
5. Return Codes: Stick to conventional meanings to Unix return codes. These are useful for chaining commands or for using in scripts. For complex failures, consider providing more detailed status information via a different mechanism (e.g., logging or a status file).
6. Security: Sanitize all inputs to prevent command injection vulnerabilities when the input is being used to construct shell command.

2.0.436 `get_ips_address`

Contained in `lib/functions.d/network-functions.sh`

Function signature: 5743d927d8ae0fb7af610c5b08a13174110679b78178a4bca8c69d9725b1fe46

2.0.437 Function Overview

The function `get_ips_address` is a simple bash function, which when executed, will output a static IP address `10.99.1.1`. This function does not receive any arguments or make use of any global variables. Its main usage is to provide the hardcoded IP as an output whenever called.

2.0.438 Technical Description

The following is a technical breakdown of the `get_ips_address` function:

- **Name:** get_ips_address
- **Description:** This function will echo the IP address 10.99.1.1 when called, without needing to provide any arguments.
- **Globals:** None
- **Arguments:** None
- **Outputs:** Prints out the static IP address `10.99.1.1`
- **Returns:** String IP address `10.99.1.1`
- **Example usage:** `get_ips_address` can be called without any arguments, simply as `get_ips_address`

2.0.439 Quality and Security Recommendations

1. Hardcoding values within a function, such as an IP address in this case, can be risky. It is generally safer and better practice to pass these as arguments or obtain them from a secure, updated source.
2. Sanity checks could be included, to assure that the output IP address is in the correct format.
3. It is recommended to use more informative function names that help other developers understand its purpose.
4. The function could benefit from handling possible errors or exceptions (eg: point to a fallback IP address or retry operation).
5. For a security improvement, the IP addresses could be stored in a secure manner, instead of being printed in plain text.

2.0.440 `get_ips_address`

Contained in `lib/functions.d/network-functions.sh`

Function signature: 5743d927d8ae0fb7af610c5b08a13174110679b78178a4bca8c69d9725b1fe46

2.0.441 Function Overview

This bash function, `get_ips_address`, essentially returns a hardcoded IP address. When called, it echoes a standard string containing an IP address. This function is pretty simple and doesn't take any arguments nor modifies any global variables.

2.0.442 Technical Description

- **Name:** get_ips_address
- **Description:** This function, returns a predefined hardcoded IP address.
- **Globals:** None
- **Arguments:** None
- **Outputs:** Echoes a string containing an IP address (10.99.1.1)
- **Returns:** Does not strictly return anything since echo is used to output the hardcoded IP address string to stdout.
- **Example usage:** `get_ips_address`

2.0.443 Quality and Security Recommendations

1. Instead of having a hardcoded value, the function should ask for input or access a configuration setting.
2. It's always a good idea to document that function does not take arguments and does not use/modify any global variables.

3. Likewise, a “returns” description could potentially indicate a limitation of the function or a place to improve upon later. You can use return statement to return value, unlike echo, this value cannot be seen, it can only be caught by capturing the exit status of the function.
4. For a more secure application, connections should ideally be made using a secure protocol, like HTTPS, that can protect the IP address from potential eavesdropping.
5. Consider how the function would behave if used in multithreaded or asynchronous processes to prevent potential race conditions.

2.0.444 `_get_iso_path`

Contained in `lib/functions.d/iso-functions.sh`

Function signature: 5a5573385030885a65194b05308aca2d42c7b55a4b3a0bb8e0435382e8d68bc5

2.0.445 Function overview

The function `_get_iso_path` is a helper function in Bash that returns a path for ISO files. It first calls another function `_get_distro_dir` which supposedly gets the directory of a particular Linux distribution. The output of this function is then concatenated with “/iso”, representing the subdirectory where the ISO files are located.

2.0.446 Technical description

- **name**: `_get_iso_path`
- **description**: This function generates the path of the directory where the ISO files of a particular Linux distribution are stored.
- **globals**: None used in this function.
- **arguments**: No arguments needed for this function.
- **outputs**: The path to the ISO directory of a specific Linux distribution, returned as a string.
- **returns**: Outputs via `echo`, the function does not explicitly return a value.
- **example usage**: The function is used as below -

  ```
  dir_path=`_get_iso_path`
  ```

  This will store the path to the iso directory in the variable `dir_path`.

2.0.447 Quality and security recommendations

1. Avoid using `echo` in functions. It can lead to problems if the output includes special characters. Use `printf` instead.

2. Don't assume the `_get_distro_dir` function will always succeed. This function should handle the case if `_get_distro_dir` fails or returns an unexpected value.
3. In general, it's good practice to handle errors and invalid usage of your Bash scripts. You could add some error handling to this function, and exit or return an error if anything goes wrong.
4. Globally declared variables can lead to unintended side effects. Avoid using them when possible.

2.0.448 `get_keysafe_dir`

Contained in `lib/functions.d/keysafe_functions.sh`

Function signature: 25a0229bfcf5422d4fe07f68f0d2261ee643e57fcff68bbc63ba33c0f41af39e

2.0.449 Function overview

The `get_keysafe_dir` function is used in the Bash shell. This function's role is to return the directory location of the keysafe within an active cluster, checking if the cluster is active, verifying if the keysafe directory exists, and creating it if it does not.

2.0.450 Technical description

- **Name**: `get_keysafe_dir`
- **Description**: The function checks if a cluster is active by checking if the symlink for the cluster directory exists. If it does not exist, an error is returned. If it does exist, the function resolves the symlink to find the actual location of the active-cluster directory and constructs the keysafe path. It then checks if the 'tokens' directory exists within the keysafe directory. If it does not, the function tries to create it. If it fails to create the directory, it returns another error. If it succeeds or if the directory already exists, it just echoes the keysafe directory path.
- **Globals**: `[ HPS_CLUSTER_CONFIG_BASE_DIR: The base directory path where the cluster configuration directories are stored ]`
- **Arguments**: The function does not take any arguments.
- **Outputs**: Prints the path to the keysafe directory or an error message.
- **Returns**: Success status of function. 0 if successful, 1 if the active cluster symlink does not exist, 2 if failed to create the keysafe directory.
- **Example usage**:

```
$ get_keysafe_dir
/path/to/your/keysafe_dir
```

2.0.451 Quality and security recommendations

1. Input Validation: Although there are no direct user inputs to this function, the paths used in the function could be validated for safety, to guard against directory traversal or similar attacks.
2. Error Handling: More detailed error messages could provide more insights about the reasons causing failure, aiding in better debugging.
3. Testing: Write test cases to cover all paths through the function especially edge cases. This includes when the symlink exists or not, and the same for the keysafe's tokens directory.
4. Avoid using globals: Usage of globals can often lead to unexpected behavior. Consider an approach where all the required parameters are passed to the function.
5. Use more secure methods for directory creation to avoid race conditions. The `mkdir -p` command can be vulnerable to race conditions, consider using `mktemp` for safer creation of directories.

2.0.452 `get_latest_alpine_version`

Contained in `lib/functions.d/tch-build.sh`

Function signature: f1404bc114d5e831d9237d9abb99e8b1e61a8b81eef840b822e47765fdfb7591

2.0.453 Function Overview

This function `get_latest_alpine_version()` is designed to fetch the latest version number of Alpine Linux from the official Alpine website. It uses either `curl` or `wget` to download the page, extracts the version number using a regular expression and sorting, and falls back to a predefined version number if it fails. If the extraction fails, the function provides a warning log message with the fallback version. Finally, the function returns this version number.

2.0.454 Technical Description

For the function `get_latest_alpine_version()`:

Name: get_latest_alpine_version

Description: This function fetches the most recent version number of Alpine Linux from its official website using either `curl` or `wget` for the process. If the methods fail in fetching, the function resorts to a fallback version defined within it. The function provides a warning log message with the fallback version in case the extraction fails and finally returns the version number.

Globals: None

Arguments: None

Outputs: - The version number of the latest Alpine Linux. - Warning message in case of failure fetching with the version falling back to 3.20.2

Returns: - 0 if the function is successful - Warning message with fallback version if the function fails

Example Usage: Run the function like so: `get_latest_alpine_version`

2.0.455 Quality and Security Recommendations

1. Error handling can be better: Currently, the function falls back to a hard-coded version when it fails to fetch the latest version via `curl` or `wget`. However, the reason might be transient network issues, and a simple retry might work. Implementing a retry mechanism with exponential backoff would improve the quality of this script.
2. Robustness: Consider checking the returned status codes of the `curl` or `wget` commands, to know if the fetch was successful or not. The result does not always indicate the command's success.
3. For security reasons, consider using a more secure protocol like secure https to access the website compared to http which is currently in use. It can protect the data being exchanged from lurking security threats.
4. The function logs a warning message if it fails to get the versions, it can also provide some debugging information like the status code, error messages, etc., which could be helpful while troubleshooting.
5. The function could have a mechanism to periodically check for an update after some interval automatically so that the fetched version is always current without the need to invoke the function manually.

2.0.456 `get_mac_from_conffile`

Contained in `lib/functions.d/network-functions.sh`

Function signature: e02347fb032c6871b64d63fdf633e3ff78088698ab33bbdb1671faa21c210ea9

2.0.457 Function Overview

This Bash function, `get_mac_from_conffile`, takes a configuration file path as an argument, and attempts to extract a MAC address from the file name. It outputs the MAC address if successful, logs an error and returns 1 if not successful.

2.0.458 Technical Description

- `name`: `get_mac_from_conffile`
- `description`: This function extracts a MAC address from the file name of a given configuration file.

- `globals`: None
- `arguments`:
  - `$1`: This is the file path of the configuration file from which the MAC address needs to be extracted.
- `outputs`: If the extraction is successful, the function echoes the MAC address to stdout.
- `returns`: The function returns 0 if extraction succeeds, and 1 otherwise.
- `example usage`:

```
MAC=$(get_mac_from_conffile
 ↪  "/path/to/conffile/abcd.efgh.ijkl.conf")
echo $MAC # outputs: abcd.efgh.ijkl
```

2.0.459 Quality and Security Recommendations

1. Validate the file path provided as an argument. Currently, the function simply checks if the argument is non-empty. It can be enhanced by checking if it is a valid path and the file exists.
2. Set a format for MAC address in the filename and validate it. The function could fail if the filename does not contain a valid MAC address.
3. Instead of suppressing errors from `basename`, handle them properly.
4. Always declare local variables at the top of the function. This makes it clear what variables are function-scoped and can prevent unexpected behavior caused by using a global variable with the same name.
5. Ensure proper and sufficient error messages are logged for better debuggability.

2.0.460 `get_mount_point_for_os`

Contained in `lib/functions.d/iso-functions.sh`

Function signature: b0f9533b66be2101b280ad32d74ddf4b9e8a4ace54c1009877962986676a7149

2.0.461 Function overview

The function `get_mount_point_for_os()` takes an `os_id` as an argument and returns the corresponding mount point for the operating system defined by the `os_id`. It first localizes the `os_id` variable to prevent it from being overwritten by subshells or other functions. The function then converts colons in the `os_id` to underscores in order to maintain filesystem compatibility. This sense the result to standard out using `echo`.

2.0.462 Technical description

- **Name:** `get_mount_point_for_os()`

- **Description:** This function converts the `os_id` argument from a colon-delimited string to an underscore-delimited string, and uses this to generate a path string that represents the mount point for the given OS in the local filesystem.
- **Globals:** None.
- **Arguments:**
  - `$1`: The `os_id` for a specific operating system. This should be a string that uniquely identifies the operating system, typically using a colon (:) as a delimiter.
- **Outputs:** Converts colons in the `os_id` to underscores (`/`), generating a mount point for the OS.
- **Returns:** A string representing the URI for the mount point within the filesystem.
- **Example Usage:**
  - `get_mount_point_for_os "Ubuntu:20.04"` might output the string `_/distro_dir/Ubuntu_20.04`.

2.0.463 Quality and security recommendations

1. Consider validating the `os_id` argument within the function to ensure it meets the expected format. This could be done using regular expressions in Bash.
2. Since this function generate a string that represents a path in the filesystem, consider validating that the resulting path actually exists in the filesystem before returning.
3. Assume that user-provided input may be malicious. It should not hurt to add checks to make sure harmful commands aren't being executed.
4. Avoid utilizing user-provided input in command lines directly to prevent command injection vulnerabilities.
5. Since the `get_mount_point_for_os()` function is written in Bash, it is crucial to avoid code that could potentially fail in different shell interpreters. It is advised to use only POSIX-compliant features of Bash.
6. Lastly, whenever we are working with file system paths, there is a potential for symbolic link (symlink) vulnerabilities. Avoid these risks by preventing the function from following symbolic links unless absolutely necessary.

2.0.464 `get_network_interfaces`

Contained in `lib/functions.d/network-functions.sh`

Function signature: 5f860b3e71cc7a8b5a1076bb49f656f0d7ddd17d0a523e3e4173aba7dffe4dc9

2.0.465 Function Overview

This Bash function, named `get_network_interfaces`, is designed to retrieve information about all the network interfaces connected to your machine excluding the localhost. For each interface, it reports the interface name, associated IPv4 CIDR notation

address, and the default gateway, if it exists. The three data points are concatenated into a string and piped through each iteration. If at least one interface is found and processed, the function returns 0, indicating success. Otherwise, it returns 1, indicating failure.

2.0.466 Technical Description

Function name: `get_network_interfaces`

Description: This function scans and retrieves information about all network interfaces on the machine, excluding localhost. The information includes interface name, IP address and gateway details for each interface.

Globals: None

Arguments: None

Outputs: Outputs a string per network interface, structured as "`interface_name`|`ip_address`|`gateway`".

Returns: `0` if at least one network interface is found and processed, `1` otherwise.

Example usage:

```
get_network_interfaces
```

The function will print to stdout each non-loopback network interface on the local machine, along with relevant information.

2.0.467 Quality and Security Recommendations

1. To reduce the risk of errors, consider validating and sanitizing command outputs before assigning them to local variables.
2. Be aware of the line `local ip_cidr=$(ip -4 -o addr show dev "$iface" 2>/dev/null | awk '{print $4}' | head -n1)`. If the `ip` command or awk is not specified correctly, the line could fail silently because of `2>/dev/null`.
3. The function currently uses a `found` variable as a flag to check if any interfaces are found. This works as expected but can potentially be refactored to address potential edge cases. For example, if the call to `ip` command fails for some reason, the function may still return 0.
4. For increased robustness, consider handling errors in a more systematic way. This means capturing and handling potential error messages from the `ip` command.
5. Be aware of potential security implications. Since this command parses command-line output, it may be vulnerable to command injection. Recommend codifying precautionary measures against such attacks.

2.0.468 `_get_os_conf_path`

Contained in `lib/functions.d/os-functions.sh`

Function signature: e2d7af556af85b02298cef2e5ba3dbd6889a3d24db96b2f309752ed6128f0bc8

2.0.469 Function overview

The function `_get_os_conf_path()` is a basic bash function that helps to generate the path of operating system configuration file - `os.conf` within a base configuration directory, named by `HPS_CONFIG_BASE`.

2.0.470 Technical description

- **Function name:**
  - `_get_os_conf_path`
- **Function description:**
  - This function generates and echoes the complete path from where configuration details related to the operating system can be fetched. It concatenates the base path pointed by `HPS_CONFIG_BASE` with the filename, `os.conf`.
- **Globals:**
  - `HPS_CONFIG_BASE: The base path to the config directory`
- **Arguments:**
  - None
- **Outputs:**
  - The absolute path to the OS configuration file
- **Returns:**
  - 0 (zero) if executed successfully
- **Example usage:**

```
source _get_os_conf_path.sh
_get_os_conf_path
```

2.0.471 Quality and security recommendations

1. This function does not handle the absence of the `HPS_CONFIG_BASE` variable. It would return `/os.conf` if `HPS_CONFIG_BASE` is not set. It is recommended to add error checking code to handle such scenarios and make the function more robust.
2. Input sanitation is not carried out. Though this function doesn't directly take user inputs, it is good to have mechanisms to safeguard against command injection vulnerabilities, especially if the content of $HPS_CONFIG_BASE is determined dynamically.
3. It's advisable to use double quotes around the variable to prevent word splitting or globbing issues, which this function already does.
4. To further enhance, you can include a help section in the function annotation providing details on what the function does, any global variables it utilizes, and an example usage.
5. To improve the overall security, consider restricting the file permissions of the script containing this function. Always follow the principle of least privilege.

2.0.472 `get_os_name`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: cf2bf59f26561392f3d83cad5ec4c7a99eaaa9641887206d46ac47a0054d2138

2.0.473 Function overview

The `get_os_name()` function is a Bash function which takes an operating system identifier as an argument and outputs the name of the operating system. This function first removes any architecture prefix in the os_id argument, then extracts the name of the operating system by discarding any part of the string after the second colon.

2.0.474 Technical description

Name:

`get_os_name`

Description:

The function returns the name part of an operating system identifier (os_id) by removing the architecture prefix and any text following the second colon.

Globals:

None.

Arguments:

- `$1`: The operating system identifier (os_id). It follows the format `arch:name:version`.

Outputs:

Outputs the name of the operating system.

Returns:

Returns nothing.

Example usage:

```
os_name=$(get_os_name "x86_64:ubuntu:18.04")
echo $os_name  # Outputs "ubuntu"
```

2.0.475 Quality and security recommendations

1. Validate the input argument to make sure it follows the expected os_id pattern before proceeding. If it does not, the function should generate an error message.
2. Quote all variable references to prevent splitting and globbing. For the `echo` command, use `printf` instead for more predictable behavior.

3. Add more inline comments to explain the purpose of the function and its logic for improved maintainability.
4. Use `unset` to clear temporary variables like `name_version` and `name` after usage for better memory management.
5. Write unit tests to automatically check the function with various input conditions to ensure it behaves as expected.

2.0.476 `get_os_name_version`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: 06c8779846fcc03db3be202c60174d8121eaf6a241f8fc0e2198feca9f802dc5

2.0.477 Function overview

The `get_os_name_version` function is essentially used in bash scripts to receive the OS (Operating System) name and version from the provided `os_id` and format it accordingly. The format is either `colon` (the default format) or `underscore`, which is specified by the user.

2.0.478 Technical description

- **name**: `get_os_name_version`
- **description**: The function utilizes the parameters `os_id` and `format` to extract the name and version of an Operating System and then format it by replacing colons with underscore if required. `os_id` is the identifier of the Operating System, which also holds the version. `format` determines how the output will be formatted; the default format is `colon`, but it can also be `underscore`.
- **globals**: None
- **arguments**:
  - `$1: os_id` - An identifier that encapsulates the name and version of an Operating System.
  - `$2: format` - Determines the format of the output. Can be `colon` (default) or `underscore`.
- **outputs**: The name and version of the operating system, formatted according to the requested format (colon or underscore).
- **returns**: Nothing (`null`)
- **example usage**: `get_os_name_version debian:11 underscore` This would return “debian_11”

2.0.479 Quality and security recommendations

1. Add validation checks for the input to ensure that the `os_id` is in a correct format and the desired format supplied is either `colon` or `underscore`. This will

prevent uncontrolled behavior with illegal inputs.
2. Consider using a more limitation-free delimiter than a colon. If the system's name or version includes a colon, this could cause erroneous output.
3. Rename the function parameters to clearly outline their purpose and role in the function. The term `format` may be unclear to some end users.
4. Include error or status messages for various steps in your script so that it helps with debugging in the future.
5. Add comment documentation for improved code readability.
6. Always keep the software and libraries used in your script up to date to minimize security vulnerabilities.
7. Ensure that the script doesn't hold any sensitive data like tokens or credentials. If it does, make sure they are securely stored and encrypted if possible.

2.0.480 `get_path_cluster_services_dir`

Contained in `lib/functions.d/system-functions.sh`

Function signature: 614369342a40a8d8eacfb7e30ad0b4a1d719139038d55af309dbc419dcd2cb3b

2.0.481 Function Overview

The function `get_path_cluster_services_dir` is designed to retrieve the path to a directory labelled 'services' within the directory of the currently active cluster. This function makes use of the `get_active_cluster_dir` function to make this determination, processes the output from this function and then appends '/services' to the end of it.

2.0.482 Technical Description

Name: `get_path_cluster_services_dir`

Description: This function generates and outputs a string that represents the path to the 'services' directory within an active cluster directory by appending '/services' to the path of the active cluster directory.

Globals: None

Arguments: None

Outputs: The function outputs a string path to a 'services' directory within an active cluster directory.

Returns: `get_path_cluster_services_dir` does not have a specific return statement, so it implicitly returns the exit status of the last command executed, which is the `echo` command.

Example Usage:

```
services_path=$(get_path_cluster_services_dir)
echo $services_path
```

2.0.483 Quality and Security Recommendations

1. Since this function relies on another function (`get_active_cluster_dir`), we must ensure that this other function has been defined physically above this function in the script file and that it works as intended.
2. Add error handling for scenarios where the `get_active_cluster_dir` function fails or returns an error to ensure that the `get_path_cluster_services_dir` function does not fail unexpectedly.
3. Consider adding the option to pass in a specific cluster rather than defaulting to the active one.
4. In the case of public scripts, remember to document all assumptions and dependencies in the script's top-level documentation, ensuring that other developers are aware of these dependencies prior to utilizing the script.
5. Lastly, always ensure that proper permissions are set for this function in order to prevent unauthorized access or modifications.

2.0.484 `get_path_cluster_services_dir`

Contained in `lib/functions.d/system-functions.sh`

Function signature: 614369342a40a8d8eacfb7e30ad0b4a1d719139038d55af309dbc419dcd2cb3b

2.0.485 Function overview

The `get_path_cluster_services_dir` is a Bash function that retrieves the path to the services directory of the currently active cluster on a system.

2.0.486 Technical description

Name: `get_path_cluster_services_dir`

Description: This function builds and displays a file path to the services directory within the currently activated cluster on a system. This is achieved by concatenating the path of the active directory, as returned by the `get_active_cluster_dir` function, with the string `/services`.

Globals: None

Arguments: None

Outputs: A string representing the file path to the services directory within the active cluster directory.

Returns: The function does not explicitly return a value, but uses the `echo` command to pass the generated path as a string to the standard output.

Example Usage:

```
services_dir=$(get_path_cluster_services_dir)
echo $services_dir
# Output: /path/to/active_cluster/services
```

2.0.487 Quality and security recommendations

1. Checking for errors: The function should check the return code of `get_active_cluster_dir` to ensure it executed successfully before proceeding to append `/services` to it. This could prevent displaying or operating on invalid paths.
2. Managing permissions: The function works with file paths which could potentially expose sensitive directories. So it should be used with care, ensuring that permissions are set correctly.
3. Logging: Consider adding logging within the function to make debugging easier in future.
4. Input Validation: Although this function does not accept arguments, for general functions that do, it's crucial to validate and sanitize the inputs.

2.0.488 `get_path_supervisord_conf`

Contained in `lib/functions.d/configure-supervisor.sh`

Function signature: 5ead4ebed48cedb7a928e3c048d0d0b546746f48cc20520b8e2cdef5bd83bbad

2.0.489 Function overview

The `get_path_supervisord_conf` function performs a simple, yet handy operation within the Bash environment. It employs the `get_path_cluster_services_dir` function to return the path of the cluster services directory. Afterwards, it appends `/supervisord.conf` to this output. The primary usage of this function is to determine the absolute path to the `supervisord.conf` file, which is vital configuration file when working with the Supervisor process controller.

2.0.490 Technical description

Definition block for `get_path_supervisord_conf` function:

- Name: `get_path_supervisord_conf`

- Description: This function constructs and returns the full path to the `supervisord.conf` file by appending its name to the output of the `get_path_cluster_services_dir` function.

- Globals: None

- Arguments: None

- Outputs: The full path to the `supervisord.conf` file.

- Returns: The output of the function is an echo command, so nothing is returned.

- Example Usage:

```
conf_path=$(get_path_supervisord_conf)
echo $conf_path
```

2.0.491 Quality and security recommendations

1. Error Handling: Currently, the function does not handle errors. It is recommended to add error handling to deal with potential problems, such as the `get_path_cluster_services_dir` function not returning a valid path.
2. Validation: Before using the output of `get_path_cluster_services_dir`, validate that the returned value is a directory that exists.
3. Secure Calls: Ensure that the `get_path_cluster_services_dir` function is defined and behaves as expected, as the `get_path_supervisord_conf` function fully relies on its output.
4. Test coverage: Include this function in unit testing to ensure its functionality doesn't break over time.
5. Return Codes: Although for this function returning a value might not be necessary or even beneficial, proper use of exit statuses can significantly improve the robustness of your scripts. It is recommended to return different status codes for different error states, which can then be used to better diagnose problems.

2.0.492 `get_path_supervisord_conf`

Contained in `lib/functions.d/configure-supervisor.sh`

Function signature: 5ead4ebed48cedb7a928e3c048d0d0b546746f48cc20520b8e2cdef5bd83bbad

2.0.493 Function overview

The function `get_path_supervisord_conf` is designed to output the path to the `supervisord.conf` file within the system's cluster services directory. This function does not take any arguments and solely depends on the functionality of another function (`get_path_cluster_services_dir`), which should ideally return the path to the cluster services directory.

2.0.494 Technical description

```
get_path_supervisord_conf () {
  echo "$(get_path_cluster_services_dir)/supervisord.conf"
}
```

Here are the technical details -

- **name**: `get_path_supervisord_conf`
- **description**: This function outputs the complete path to the `supervisord.conf` file in the cluster services directory. It echo's the result of `get_path_cluster_services_dir` function call concatenated with '/supervisord.conf'.
- **globals**: None.
- **arguments**: None.
- **outputs**: Prints the path to the `supervisord.conf` file.
- **returns**: None, as the function result is directly printed out.
- **example usage**:

```
get_path_supervisord_conf
```

2.0.495 Quality and security recommendations

1. Make sure the `get_path_cluster_services_dir` function is safe and secure. This function's output is directly used here, so any security vulnerabilities in it might affect this function too.
2. The function does not handle the case where the `get_path_cluster_services_dir` might fail or return an error. It is recommended to introduce some error checking to make the function more reliable.
3. The function does not check whether the `supervisord.conf` file is reachable or accessible. Adding this check will improve the reliability of the function.
4. The function does not validate the path string it constructs, making it potentially vulnerable to path traversal or other filesystem-based attacks. Validate and sanitize all outputs from `get_path_cluster_services_dir` function.
5. Ensure the service (or script) that uses the output of this function has proper permissions to access the `supervisord.conf` file. This will prevent possible permission issues at runtime.

2.0.496 `get_tch_apkovl_filename`

Contained in `lib/functions.d/alpine-tch-build.sh`

Function signature: 4373c08348b4aa83cad647311c581a9a733b25a751bcadc17ade3e1a6d42169d

2.0.497 1. Function Overview

The Bash function `get_tch_apkovl_filename()` is a simple utility function. The primary purpose of this function is to echo out a specific static string: "tch-bootstrap.apkovl.tar.gz".

2.0.498 2. Technical Description

- **name**: `get_tch_apkovl_filename`

- **description**: This is a utility function that does not take any variables or arguments but outputs a static string “tch-bootstrap.apkovl.tar.gz”. This function can be used in scripts where this string is repeatedly needed.
- **globals**: None
- **arguments**: None
- **outputs**: The static string “tch-bootstrap.apkovl.tar.gz”.
- **returns**: The function does not return any result aside from the output string.
- **example usage**: The function can be used in any script by simply calling it. Here’s a basic usage example:

```
$ get_tch_apkovl_filename
```

2.0.499 3. Quality and Security Recommendations

1. Given that this function simply echoes a static string, there is little scope for security vulnerabilities. However, if you plan to extend the functionality, consider implementing input validation to ensure only expected arguments are passed.
2. If the output string is liable to change, consider parameterizing this function in order to make it more adaptable and less prone to require edits within its body.
3. Under potential refactorings or enhancements, always consider employing test-driven development practices. Create unit tests for different scenarios, to ensure that function works as expected.
4. Code documentation: Even though the function is simple, it is a good practice to describe what exactly it does in a comment just above the function definition.

2.0.500 `get_tch_apkovl_filename`

Contained in `lib/functions.d/alpine-tch-build.sh`

Function signature: 4373c08348b4aa83cad647311c581a9a733b25a751bcadc17ade3e1a6d42169d

2.0.501 1. Function Overview

The `get_tch_apkovl_filename` function in the Bash script, when executed, doesn’t take any parameters and has no internal workings. It just outputs a predefined string “tch-bootstrap.apkovl.tar.gz”. This function serves its purpose when there is need to obtain a specific filename in different parts of an application thereby avoiding hardcoding of strings and potential typos.

2.0.502 2. Technical Description

- **Name:** get_tch_apkovl_filename

- **Description:** This function returns a pre-defined string “tch-bootstrap.apkovl.tar.gz”. It does not perform any operation.
- **Globals:** None
- **Arguments:** None
- **Outputs:** A hardcoded string - “tch-bootstrap.apkovl.tar.gz”.
- **Returns:** The string “tch-bootstrap.apkovl.tar.gz”.
- **Example Usage:** The function will be called as `get_tch_apkovl_filename`. It does not require any arguments. The return value could be assigned to a variable like so: `filename=$(get_tch_apkovl_filename)`.

2.0.503 3. Quality and Security Recommendations

1. Currently, the function is simple and secure, as there’s no interaction with user input or external data.
2. In the future, if it is supposed to return variable data, consider input validation to ensure safety and prevent potential code injection.
3. Always perform testing on code functions to ensure they’re performing as expected.
4. If possible, add error handling methods within the function to avoid possible execution halt or unexpected system errors.
5. Comment your code: Although this function is simple, good practice is to maintain a habit of documenting what a function does, its inputs, its outputs, and any global variables it interacts with.

2.0.504 `get_tch_apkovl_filepath`

Contained in `lib/functions.d/alpine-tch-build.sh`

Function signature: 12c2f4d9ccb7cebc830cde3dcc920a49c1633b192c54ce1b37d9e4941a1043a5

2.0.505 Function overview

The bash function `get_tch_apkovl_filepath()` is designed to accept an operating system ID as its argument and generate a file path for the overlay file of that specific operating system. The function first checks if the configuration for the given operating system ID exists using `os_config` function. If it doesn’t exist, it logs an error and returns exit code 1. If it does exist, it gets the repository path and combines with the overlay file name to produce the full file path.

2.0.506 Technical description

- **Name:** `get_tch_apkovl_filepath()`
- **Description:** This function generates and prints the file path for the overlay file of a certain operating system based on the provided operating system ID.

- **Globals:** None.
- **Arguments:**
  - `$1`: This is a string that represents the operating system ID. The function will fail with a usage error if this argument is not provided. It is used to query the os configuration and to log an error message if the os configuration doesn't exist.
- **Outputs:** This function prints the path for the overlay file of the given operating system to stdout.
- **Returns:** The function returns exit code 1 if the os configuration for the given os ID doesn't exist. Otherwise, it returns success (exit code 0).
- **Example usage:** `get_tch_apkovl_filepath ubuntu18`

2.0.507 Quality and security recommendations

1. The function should validate the provided OS ID before using it to prevent potential command injection vulnerabilities.
2. It would be advisable to check whether the generated file path exists before returning it. If the file or directory doesn't exist, the function should log an error and return a non-zero exit code.
3. The function depends on other external functions like `os_config` and `get_tch_apkovl_filename`. It should check whether these functions exist before calling them to prevent potential command not found errors.
4. It would be prudent to use double quotes around variable references to prevent word splitting and globbing issues. For instance, it's recommended to use quotes around `"$(_get_distro_dir)/${repo_path}/$(get_tch_apkovl_filename)"`.

2.0.508 `__guard_source`

Contained in `lib/functions.sh`

Function signature: e8beb9b32cabb9e73dba64f7b102ad5f1112590b455a914144bd65e5d3001168

2.0.509 Function overview

The function `__guard_source()` is a bash script guard, designed to prevent sourcing a script more than once. This function takes the name of a previously-sourced script and stores it in a guard variable. The function returns '1' if the script has already been sourced (indicating an error and preventing further sourcing) and '0' if not.

2.0.510 Technical description

- **name:** `__guard_source()`

- **description:** This function is used to avoid multiple sourcing of the same bash script for the prevention of redundant operations, potential recursion and unexpected behavior. It generates a guard variable for the sourced script and checks if the script has been previously executed or sourced, and if so, the function will return 1 preventing the script from re-executing.
- **globals:** `[_guard_var: Guard variable created for each sourced script to mark its execution. It uses the script name with nonalphanumeric replaced with '_']`
- **arguments:** `[$1: Unused in this function.,$2: Also unused in this function.]`
- **outputs:** No output is returned to stdout/stderr.
- **returns:** `0 if the script or source has not been run before, and 1 if it has and encoutered an error`
- **example usage:**

```
if ! __guard_source; then
    echo "Script already sourced once"
fi
source script.sh
```

2.0.511 Quality and security recommendations

1. Clear documentation: Each function, global variable and return value should be clearly documented for future developers.
2. Error handling: The function should handle possible errors gracefully and clear, meaningful messages should be returned.
3. Input Validation: Currently, the function does not take any arguments, but for future enhancement if it does, it should validate the input before processing it.
4. Use Strict Mode: To catch errors and undefined variables sooner, consider enabling a 'strict mode' in your scripts with `set -euo pipefail`.

2.0.512 `handle_menu_item`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: ae91eecf756179bb9ef9c963e51207d99a2a60ff1a8c1f3a588c7faa1749c58b

2.0.513 Function overview

The function `handle_menu_item()` is an efficient way to manage any ipxe menu function across all menus. It accepts two arguments and, based on the first one, it performs a variety of tasks like initialization, configuration, displaying information, system reboots, running Boot installer, and rescue. It also allows toggling Force Install and displays a fail message for any unknown menu items.

2.0.514 Technical description

Here's a detailed technical description for the `handle_menu_item()` function:

- **Name**: `handle_menu_item`
- **Description**: A function that handles an ipxe menu function across all menus.
- **Globals**: Not applicable.
- **Arguments**:
  - `$1`: This represents the menu item to be handled.
  - `$2`: This refers to the Media Access Control (MAC) address.
- **Outputs**: The function outputs a series of events according to the condition met by the argument `$1`.
- **Returns**: It may return `1` if an invalid install item format is encountered.
- **Example Usage**:
  - `handle_menu_item init_menu AA:BB:CC:DD:EE:FF`
  - `handle_menu_item host_configure_menu AA:BB:CC:DD:EE:FF`

2.0.515 Quality and security recommendations

Here are some quality and security recommendations for the `handle_menu_item` function:

1. Always validate the inputs beforehand. The function does not validate or sanitize the inputs `$1` and `$2` which can potentially lead to security vulnerabilities.
2. Document not only the function itself but also each of its components to improve maintainability.
3. Consider handling an error explicitly when the case statement does not match any condition. Currently, it outputs a log.
4. Test the function with various inputs to ensure its robustness and resilience.
5. Evaluate the use of global functions or variables within the function for potential conflicts or issues. Keep their scope as small as possible.
6. Consider returning distinct error codes for different error cases to allow the caller to create more detailed error handling procedures.

2.0.516 `has_sch_host`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 8239c1c521d6e5ec4f193188027bf12f8f78909eebda3b607d1d03cc1835a65d

2.0.517 Function overview

The `has_sch_host` function is a bash function designed to check whether there is at least one SCH host present using the cluster helper function with type filter. If there is at least one SCH host, the function returns `0`. If there is not, the function returns `1`.

2.0.518 Technical description

Name: has_sch_host
Description: This function is used to obtain all SCH hosts using the `get_cluster_host_hostnames` function. It checks if there is any SCH host and returns 0 or 1 based on the outcome.
Globals: None
Arguments: None
Outputs: This function does not directly output to stdout. However, it does store the list of SCH hosts in a local variable.
Returns:
- If there are SCH hosts, it returns 0.
- If there are no SCH hosts, it returns 1.

Example Usage:

```
if has_sch_host; then
  echo "There is at least one SCH host."
else
  echo "There are no SCH hosts."
fi
```

2.0.519 Quality and security recommendations

1. Input validation: Always validate input passed into the `get_cluster_host_hostnames` function. Even though it's not directly user input, it's always a good practice to validate input.
2. Error handling: Consider including error handling measures for the scenario when the `get_cluster_host_hostnames` function fails to execute.
3. Security: If the `get_cluster_host_hostnames` function is handling sensitive information, consider integrating security measures to protect this data.
4. Return Usage: Continue to use return codes to indicate state, as it follows expected Unix practices.
5. Commenting: More comments can be added for each line describing what each command does. This will help other developers to understand your code quickly.

2.0.520 `host_config_delete`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 879c4c1c59478ffcfd437a6d710d463fd2811a08a4bc4f57d5dd6d62b5a37709

2.0.521 Function Overview

The `host_config_delete` function is designed to validate and delete configuration file associated with a provided MAC address. It uses a helper function

`get_host_conf_filename` to determine the config file path. If the MAC address or config file are not found, or if the file deletion fails, it logs an error and returns an appropriate error code.

2.0.522 Technical Description

Name:
`host_config_delete`

Description:
This function deletes the configuration file for a provided MAC address. It logs error messages in case the MAC address is not provided, config file location cannot be determined, and deletion fails.

Globals:
- `__HOST_CONFIG_MAC`: Stores the last valid MAC address
- `__HOST_CONFIG_PARSED`: Indicates if parsed files are available
- `__HOST_CONFIG_FILE`: Stores host_config file path

Arguments:
- $1: The MAC address

Outputs:
Logs messages to indicate success or failure of operations.

Returns:
- 0: If deletion is successful. - 1: If MAC address is not provided or config file location cannot be determined. - 2: If the config file doesn't exist. - 3: If deletion fails.

Example usage:

```
host_config_delete "00:14:22:01:23:45"
```

2.0.523 Quality and Security Recommendations

1. Improve error logging framework: The error messages could be improved by providing more details about the failures.
2. Properly handle globals: Globals can lead to hard-to-trace bugs and reduce testability. Avoid globals when possible, and instead use function returns and arguments.
3. Potential race conditions: As the function checks for file existence and then deletes it, there could be a race condition if another process deletes the file after the check but before the delete. To prevent this, the function could catch and handle the error case where the file doesn't exist at removal stage, rather than checking file existence in advance.
4. Return comprehensive status codes: A dedicated error code should be provided for each unique error case to enhance maintainability and troubleshooting.

2.0.524 host_config_exists

Contained in `lib/functions.d/host-functions.sh`

Function signature: 34de36858dffe5d99bd23615eca3110b87f3a65c04849bd3f82c9ba9fb2194ea

2.0.525 Function Overview

`host_config_exists` is a bash function that is used to check if the configuration file for a particular host (identified by its MAC address) exists and is readable.

2.0.526 Technical Description

- **Name:** `host_config_exists`
- **Description:** It accepts a MAC address as an argument and verifies if a corresponding configuration file exists on the local system. To ensure the check is reliable, the function employs `get_host_conf_filename`, which performs the actual existence and readability checks.
- **Globals:** None
- **Arguments:**
  - `$1`: MAC address (required) - The MAC address of the host for which the configuration file's existence is being checked.
- **Outputs:** This function does not output anything to STDOUT; it only uses exit codes to indicate results.
- **Returns:** Returns are exit codes and have the following meanings;
  - `0` - The configuration file exists and is readable.
  - `1` - No MAC address provided or the configuration file does not exist or is not readable.
- **Example Usage:**

```
if host_config_exists "01:23:45:67:89:ab"; then
  echo "Configuration exists for host"
else
  echo "Configuration does not exist for host"
fi
```

2.0.527 Quality and Security Recommendations

1. Always validate externally supplied variables before using them in the function.
2. Consider logging error messages in case of an exception or an unexpected scenario.
3. Remember to keep config files secured with correct permissions to avoid unauthorized access.
4. Always keep the function up to date considering any changes made in the `get_host_conf_filename` function since it relies on it.
5. Avoid using globals as they may produce unpredictable results, thus reducing the function's reusability.

2.0.528 host_config

Contained in lib/functions.d/host-functions.sh

Function signature: 1c42f4e63427561938e597d7d00710a1c304379c69126407f57b12c51434224a

2.0.529 Function Overview

The host_config function in Bash is used to handle configuration settings related to a host. Identified by its MAC address, the host's configuration is stored within a file. The function facilitates getting, setting, checking existence, equality, and deletion of configuration keys and values. The MAC addresses are validated and normalized for processing. Moreover, it utilizes local variables to prevent value collision.

2.0.530 Technical Description

- **name**: host_config
- **description**: A Bash function to manage a host's configuration settings via a file identified by the host's MAC address. Operation modes include getting, setting, checking existence, equality, and deletion of configuration keys and values.
- **globals**: None
- **arguments**:
  - $1: mac_param: MAC address of the host
  - $2: cmd: Command to perform (get, set, exits, equals, unset)
  - $3: key: Configuration key
  - $4: value: Value to set for a configuration key (only required for set command)
- **outputs**: Varies depending on the command being executed. May output configuration value, success or error messages.
- **returns**:
  - 0 on successfully getting, setting, or deleting a configuration key
  - 1 on MAC address validation failure, failure to normalize MAC address, or failure to determine active cluster hosts directory
  - 2 on invalid command or invalid key format
  - 3 on failure to create configuration directory or failure to write to configuration file
- **example usage**: host_Config "00:0a:95:9d:68:16" "get" "config_key"

2.0.531 Quality and Security Recommendations

1. All user inputs including MAC address, key, and value should be validated thoroughly to prevent any form of invalid data or injection attacks.

2. Error messages need to be descriptive yet should not reveal too much about the system structure or file locations fending off potential security risks.
3. Keep updating and getting values atomic. Ensure that any update operation is fully completed before another read or update operation is started.
4. Always make sure that the privileges for creating directories and files are minimal and do not give unnecessary permissions.
5. Incorporate logging for every major step inside the function, especially for the operations changing the system state for ease of troubleshooting and potential auditing purposes.
6. Error handling mechanism would be improved by differentiating error types and handling them individually. Hence a standard error explanatory mechanism will help users understand why a particular error occurred.

2.0.532 `host_config_show`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 84f65b3dca74b99c5f2890c67f216c67c0deb77fe856d7ec3d6f8447b05622e8

2.0.533 Function Overview

The `host_config_show()` function in Bash is used to display the configuration of a specific host, based on the provided MAC address. After validating the MAC is provided, it uses a helper function to get the path of the config file, perform necessary checks, and then it reads and displays the config file while managing the format, escape sequences, and quotes. If the config file is not found, it throws an information log entry with the respective MAC.

2.0.534 Technical Description

- **Name:** host_config_show
- **Description:** This bash function displays the configuration of a specified host based on the MAC address supplied as an argument. It checks for the validity and existence of the MAC and reads the corresponding configuration file. Comments and empty lines in this file are skipped and in the end, key-value pairs are displayed in a specifically formatted manner.
- **Globals:** N/A
- **Arguments:**
    1. `$1` - The MAC address. It must be supplied for the function to display the configuration.
- **Outputs:** The function outputs key-value pairs from a config file. Formatted as `key="value"`.
- **Returns:** The function returns 0 if the operations are successful and the key-value pairs have been displayed; 1 if a MAC address is not provided or not found, in these

cases an info log or error log is displayed.
- **Example usage:** `host_config_show "00:0a:95:9d:68:16"`

2.0.535 Quality and Security Recommendations

1. The function can be improved by making sure that the MAC address passed is in the valid format. This can be achieved by using regular expressions.
2. The error messages can be made more detailed to explain the underlying issues more clearly.
3. Consider handling the reading of the file line by line in a more safe manner for experimental file types.
4. Handle exceptions and edge cases, such as non-string inputs or unexpected behavior from the helper function `get_host_conf_filename`.
5. For better security, always sanitize the provided input to prevent potential Bash injection attacks.

2.0.536 `host_initialise_config`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 4bfc4920ed7de916252ca536f0799188fb42d5a0f55084a1d8ff3a64fdee3cb5

2.0.537 Function overview

The `host_initialise_config` function is primarily used to delegate host configuration in a server network by assigning network configurations based on the MAC address. It works by first validating if the required MAC address is provided, checks if the relevant directory exists for the active cluster, creates it if it doesn't, and then proceeds to set the initial state and architecture.

2.0.538 Technical description

Definition block for `host_initialise_config`:

- **name**: `host_initialise_config`
- **description**: The function initialises network configuration for an incoming server based on its MAC address. Prepares the host directory and sets up the initial state to UNCONFIGURED and assigns the architecture.
- **globals**: None
- **arguments**: `$1: MAC address of the new host`, `$2: Architecture of the new host`
- **outputs**: Error messages in case the MAC address is not provided, the active cluster host directory cannot be determined, failure in creating the hosts directory, or in initializing the state and architecture for the new host.

- **returns**: 1 if any error is encountered, otherwise 0 after successfully initializing the configuration.
- **example usage**: `host_initialise_config 0242ac120003 x86_64`

2.0.539 Quality and security recommendations

1. Improve error handling by making messages more descriptive and therefore more helpful for debugging purposes.
2. Consider using more specific validation rules for MAC addresses and architecture input.
3. Verify the permissions for directories and files involved; ensure they are locked down to prevent unauthorised access or modifications.
4. Implement logging of successful operations for auditing purposes.
5. Involve error codes alongside messages to provide an easier way to manage or handle errors.

2.0.540 `host_network_configure`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 7bcf1eba347af3f76fd003ebc949da7f788e50ef535159cf9a85a738271674c7

2.0.541 Function Overview

The function `host_network_configure` configures the network settings for a given host in a cluster by validating input parameters, fetching and validating the network configuration of the cluster, preserving or creating new IPs and hostnames, and writing all of this information to a network configuration.

2.0.542 Technical Description

- **Name**: `host_network_configure`
- **Description**: This function configures network settings for a host based on its MAC Address and host type.
- **Globals**: None
- **Arguments**:
  - `$1`: `macid` (MAC Address ID)
  - `$2`: `hosttype` (Type of host)
- **Outputs**: Logs detailing the process, and error messages if any error occur.
- **Returns**:
  - `1` if any error occurs. For example, missing MAC Address, missing or invalid network configuration, insufficient IPs in the DHCP range, and inability to generate a valid hostname.
  - `0` if the network configuration is successful.

- **Example Usage**: `host_network_configure "00:14:22:01:23:45" "guest"`

2.0.543 Quality and Security Recommendations

1. In order to enhance the quality of this function, consider adding more detailed logging that could help troubleshooting potential issues more conveniently.
2. It may be advisable to include non-zero return codes for every specific error to differentiate between different error cases.
3. Incorporate further defensive programming measures, such as input sanitization, to prevent potential security vulnerabilities.
4. Avoid logging sensitive information such as MAC Addresses and IPs to comply with security best practices.
5. Consider implementing error checking and retry logic where appropriate to increase the robustness of network configurations.

2.0.544 `host_post_config_hooks`

Contained in `lib/functions.d/host-functions.sh`

Function signature: efb256e638c74f31eec111be92bae49216e5d4abc1cabf2c6961ab1b52ec20b7

2.0.545 Function Overview

The `host_post_config_hooks` function in bash is used to detect whether a given key has been assigned a hook in a specified array of hooks. If it has, the function checks if the hook function exists and then calls it, logging this process regardless of the function's existence or possible failures. It always returns success, denoted by '0' in bash.

2.0.546 Technical Description

- **Name:** `host_post_config_hooks`
- **Description:** This function takes two inputs, a key and a value. It first defines a list of hooks and checks if the key has an associated hook. If such a hook exists, it verifies if the function for the hook exists before calling it. If it fails or the function doesn't exist, it logs the incident.
- **Globals:** No global variables are used.
- **Arguments:**
  - `$1: key` - The identifier for the hook to be invoked.
  - `$2: value` - This value is only referenced, but not used within the function.
- **Outputs:** Log messages informing about the call status and any incidents occurred.
- **Returns:** The function always returns success, represented by '0' in bash, regardless of whether the hook function exists or runs successfully.

- **Example usage:** `host_post_config_hooks "IP" "192.168.1.2"`

2.0.547 Quality and Security Recommendations

1. **Error Handling:** Function can be improved by better error handling. The value variable $2 should be utilized, especially when the hook functions are failing.
2. **Failure Response:** Always returning success regardless of function failure could cause downstream errors to be undetected. Consider returning different codes for differing process states.
3. **Function Validation:** Validate input parameters to ensure key-value pairs being processed are expected system variables.
4. **Logging Levels:** Implement more granular logging levels, providing more detailed error messages in case of a failure. It will help in debugging cases when hook function fails.
5. **Security:** Be aware of possible command injections via the key-value parameters; ensure proper sanitization or escape of possible control characters.
6. **Usage of Globals:** Consider passing all required data as function arguments, rather than relying on global variables, to decrease dependencies and improve the portability and testability of the function.

2.0.548 `hps_check_bash_syntax`

Contained in `lib/functions-core-lib.sh`

Function signature: 2d9f400c53db16cd63c0ddf7dae99ded12be73f50541543cbe59e4c48e7d1fe5

2.0.549 Function overview

`hps_check_bash_syntax` is a Bash function responsible for checking the syntax of a specified bash code. When invoked, it receives input as a string of Bash code or a filename, verifies its correctness with a Bash syntax check, and outputs any syntax errors detected. The function also provides plenty of informative feedback, such as code context, function name, and helpful hints for addressing active error types.

2.0.550 Technical description

- **Name:** `hps_check_bash_syntax`
- **Description:** Checks the syntax of a given Bash code or a file containing the Bash code. It also outputs the detected syntax errors.
- **Globals:** VAR is not clear from the given code.
- **Arguments:**
  - `$1: input`. The first argument represents either Bash code or a filename comprising the Bash code.

  - `$2: label`. The second argument is a label used in output messages for clarity.
- **Outputs:** On successful syntax check, it prints the log message "[SYNTAX] ✓ Syntax check passed for $label". If any syntax errors are found, it prints an elaborate report for each error containing error message, error line number, function causing the error, and a context view (5 lines before and after the error).
- **Returns:**
  - `0` when the syntax check is successful.
  - `1` when the syntax check fails.
- **Example usage:**

```
hps_check_bash_syntax "foo.sh" "Foo script"
```

2.0.551 Quality and security recommendations

1. Where possible, avoid the use of temporary files in `/tmp` as they could be vulnerable to symlink attacks. Consider using a more secure way to create temporary files such as `mktemp`.
2. Refrain from revealing too much information about syntax errors as this could expose sensitive details about the internal workings of the script, which malicious users can potentially exploit.
3. Make sure all input data, especially if it's coming from an external source, is validated and sanitized to protect against injection attacks.
4. Consider providing an option to silence the output, as verbose output might flood the terminal screen in a large project.
5. Always consider input case sensitivity as an important factor while performing checks on input.
6. Save and restore the state of any global variables used to prevent side effects on the rest of the program.
7. Error messages should consistently be sent to `stderr` to avoid confusion and maintain the separation between regular output and error messages. Regular output should be sent to `stdout`.

2.0.552 `hps_debug_function_load`

Contained in `lib/functions-core-lib.sh`

Function signature: f1f5c3ace74ef5ae34b33dba1dc07dcd437dd168d1a336c6e25c4dc0856c8428

2.0.553 Function overview

The function `hps_debug_function_load` is used for debugging Bash functions by providing an analysis of the given input, which can either be a Bash function or a function file. The function performs a basic syntax check, lists all functions within the input, tests

individual function loads, and also attempts a full file load of the function(s). In case of any syntax or load errors, related debug information will be provided.

2.0.554 Technical description

- **Name**: `hps_debug_function_load`
- **Description**: This function is used to debug and analyze both individual Bash functions and function files. It lists all the functions found in the input, checks their syntax, loads the functions individually, and attempts a full file load. If the input is "-", it is considered as a file and handled appropriately.
- **Globals**: None
- **Arguments**:
  - `$1: input`: This is the input to be analyzed. It can be a Bash function or a function file. If the input is "-", it is treated as a function file.
  - `$2: label`: This is the given label for the provided input. The default label is "function file".
- **Outputs**: Debug information related to syntax checks, functions found, individual function loads, and full file load.
- **Returns**: The function returns 1 if there are syntax errors or if not all functions load successfully, and 0 when all checks and the full file load are successful.
- **Example Usage**:

```
hps_debug_function_load "/path/to/function/file" "Custom
 ↪  Label"
```

2.0.555 Quality and security recommendations

1. Always sanitize user input especially if the function is exposed to users directly.
2. Incorporate logging into the function to record every debugging information for future analysis and potential audits.
3. Use more explicit ways to handle errors by providing more descriptive error messages.
4. Implement restrictions on the kind and size of files that can be tested.
5. Consider using a unique temporary file for each instance of the function in order to prevent possible race conditions.
6. Always clean up temporary files even when an error occurs in the function to prevent buildup of residual files.
7. Use exit codes that are more descriptive for ease of debugging and maintaining the function in the future.
8. Don't use hardcoded paths for file read/write, instead use configurable paths.

2.0.556 `hps_find_syntax_pattern`

Contained in `lib/functions-core-lib.sh`

Function signature: b0d91ff68fd9a6a24a419623f8927ffce53585093ba2104491e9de28c3f63b96

2.0.557 Function Overview

The `hps_find_syntax_pattern` function is designed to analyze a specified file and recognize potential issues in the Bash script. It achieves this by checking for various common errors such as unmatched quotes, parentheses mismatches and structural issues around the use of if/then/fi. The function prints analysis details to the standard error output to ensure they will not interfere with the standard output of script execution.

2.0.558 Technical Description

Function name hps_find_syntax_pattern

Description This function is designed for searching common issues related to syntax in Bash files including unmatched quotes and parentheses, also structural issues in using if/then/fi.

Globals None

Arguments - `$1`: The file path to be checked for syntax issues. - `$2`: The specific line number in the provided file where it stops checking for common issues.

Outputs This function prints out the findings of the analysis including details about recent quotes, parentheses mismatches and whether there is a missing “then” after an “if” to the standard error.

Return None

Usage

```
hps_find_syntax_pattern "test_file.sh" "150"
```

2.0.559 Quality and Security Recommendations

1. Enhance error handling: Currently the function does not handle errors that might occur if the provided file is not found or insufficient permissions are present to read the file. Consider adding error handling for these scenarios.
2. Validate inputs: The function assumes that the second argument is always a number. Add validation to handle scenarios where the second argument can be a non-numeric value or zero.
3. Sanitize inputs: Before working with user-provided inputs consider sanitizing it to prevent potential security issues such as path traversal or command injection. For example, ensure that the provided file path does not contain harmful sequences like `../`.

4. Handle edge cases: If the “if” count doesn’t equal the “then” count, the function suggests that there may be missing “then” directives but this might not always be accurate. For example, the “if” directive could indeed be missing from a “then…fi” structure. Update your function to handle these edge cases accurately for more precise analysis.

2.0.560 hps_get_distro_string

Contained in `lib/functions.d/node-bootstrap-functions.sh`

Function signature: d0a42fae7fd315ef37bbda7684d00b2b35c3ca46bc5849dfbcf8e045d9686609

2.0.561 Function Overview

The `hps_get_distro_string` function generates a string in the format of `cpu-manufacturer-osname-osversion`. It first identifies the CPU architecture and uses “linux” as the hardcoded manufacturer. Then it checks if the `/etc/os-release` file exists.

If the file exists, it sources the file to acquire operating system name (`osname`) and version (`osver`). If the `/etc/os-release` file does not exist, it set `osname` and `osver` to “unknown”. The function then prints the constructed string.

2.0.562 Technical Description

- **name**: `hps_get_distro_string`
- **description**: Generates a string that contains the system’s CPU architecture, hard-coded manufacturer “linux”, operating system name, and version. The format is `cpu-manufacturer-osname-osversion`.
- **globals**: None
- **arguments**: None
- **outputs**: A string with the format `cpu-manufacturer-osname-osversion`. For example, `x86_64-linux-ubuntu-20.04`.
- **returns**: None
- **example usage**:

```
distro_info=$(hps_get_distro_string)
echo $distro_info
```

2.0.563 Quality and Security Recommendations

1. Validate the platform before running: This function is highly dependent on the structure and existence of `/etc/os-release`. On platforms where this file does not exist or is formatted differently, the function may not behave as expected.
2. Variable sanitization: Make sure that the variables `cpu`, `osname` and `osver` do not contain harmful characters, which could lead to command injection vulnerabilities.

3. Independent from variable scope: The script uses a shell dot (.) to include another file into the script (/etc/os-release). This could potentially alter other variables in the scope. Instead, read the values without altering the environment.

2.0.564 hps_get_provisioning_node

Contained in `lib/functions.d/node-bootstrap-functions.sh`

Function signature: fc1fca1d0e398611b360f9b260e59bc31da15d0f991d4078b185ae2bb7994011

2.0.565 1. Function overview

The `hps_get_provisioning_node()` function is a bash utility that is intended to retrieve the default gateway IP in a given machine's network routes. This function may serve purposes such as network diagnostics or configuration, and is generally utilized in Unix environments. This function using `ip route` command to get the route list and awk command to print the third column of the first line that starts with 'default', which is the default gateway IP.

2.0.566 2. Technical description

- **Name**: `hps_get_provisioning_node`
- **Description**: This is a bash function designed to get the IP of the default gateway. It uses `ip route` to get the list of network routes and passes the output to awk which prints the third field of the first line that starts with 'default'. The function immediately exits after printing to avoid processing the rest of the routes.
- **Globals**: None.
- **Arguments**: None.
- **Outputs**: The IP address of the default gateway.
- **Returns**: The exit status of the awk command, which is 0 if the command executes successfully and an error code otherwise.
- **Example usage**:

```
GATEWAY_NODE=$(hps_get_provisioning_node)
echo $GATEWAY_NODE
```

2.0.567 3. Quality and security recommendations

1. The function currently doesn't handle errors or unexpected situations. It would be wise to include error handling and exit codes indicating the type of error.
2. Although the usage of awk here is efficient and secure, misuse of external commands can lead to security vulnerabilities.
3. Use full paths to programs to avoid hijacking. For example, instead of `ip route`, you might want to use `/sbin/ip route`.

4. Validate the output before using it. In this case, confirm that the output is a valid IP address. Validation can protect against misinterpretation of erroneous outputs.
5. Geared towards Unix systems, function compatibility with non-Unix environments should be considered for improvements.

2.0.568 hps_get_remote_functions

Contained in `lib/functions.d/node-libraries-init.sh`

Function signature: 68497d8963533f0a35241d2521d849c9650a32610e875bba4872770a0590e55c

2.0.569 Function overview

The `hps_get_remote_functions` bash function retrieves relevant configuration details for a specified host. The function starts by fetching the `os_id` for the host using the MAC address provided. If unsuccessful, the function logs an error message and returns 1. Then, it retrieves the values for type, profile, state, and rescue of the host. If any of these not found, default values are used. After logging these details, the function generates an output function bundle by calling `node_build_functions` with the retrieved or default values as arguments.

2.0.570 Technical description

2.0.570.1 Function Definition:

- **name**: `hps_get_remote_functions`
- **description**: This function retrieves the host configuration details (os_id, type, profile, state, rescue) based on the given MAC address. It logs an error if os_id is not found or is empty, returns 1 in such cases. At the end, it calls `node_build_functions` with found or default values and eventually, returns 0.
- **globals**: [mac: Host's MAC address]
- **arguments**: [$os_id: Operating System ID, $type: host type, $profile: host profile, $state: host state, $rescue: boolean indicating rescue mode]
- **outputs**: The function logs error, info messages and outputs function bundle if all goes well.
- **returns**: The function returns 0 after successful execution, returns 1 when os_id is not found or empty.
- **example usage**: `hps_get_remote_functions`

2.0.571 Quality and security recommendations

1. Validation checks for the variables before proceeding to operations could be added. This will ensure that we have valid data before proceeding to the next step.

2. The function should handle exceptions and/or error scenarios more gracefully. It currently returns after logging the error but it would be more robust if it had a recovery or an alternate path.
3. Providing sensitive data like MAC address directly in function arguments might pose a security risk. It's better to prompt user for input or retrieve from a secure source at runtime rather than having it hardcoded or directly passed.
4. Error messages should be made more informative. In current scenario, only basic output is given which might not be helpful in debugging scenarios.
5. Abstraction and modularization could be improved, the function seems to be doing a lot of things. Splitting it into different functions would improve readability and maintainability.
6. It's important to follow a consistent naming convention for variables and functions. This helps in understanding the code better and quicker especially during maintenance.

2.0.572 `hps_load_node_functions`

Contained in `lib/functions.d/node-bootstrap-functions.sh`

Function signature: d293323e05ffb639ae7859eabffd99f5e9aa109ae0cdeba1581468355460151c

2.0.573 Function Overview

The `hps_load_node_functions` function is part of a more substantial script, intended for loading additional functions from a remote server identified as IPS. Communication with this server is made via the HTTP protocol. If the function fails to download the required data from the server or encounters other issues, such as problems with the downloaded data's evaluation, it falls back to a locally cached version of these functions.

2.0.574 Technical Description

- **Name:** `hps_load_node_functions`
- **Description:** This function loads functions from a remote server. If this operation fails, it attempts to load a cached version of these functions. The function determines the provisioning node, builds a URL, fetches functions from the server, validates the response, caches the response and sources them into the current shell.
- **Globals:** None
- **Arguments:** None
- **Outputs:** Logs statements on stderr about the loading operations' status and potential errors.

- **Returns:** `0` if the functions were loaded successfully or loaded from the cache. Returns `1` if it could not determine the provisioning node or if it failed to evaluate the functions. Returns 2 if failed to source cached functions or no cache is available, or when there's an empty response from the IPS.
- **Example Usage:**

  ```
  hps_load_node_functions
  ```

2.0.575 Quality and Security Recommendations

1. Use more specific error codes to differentiate between various error conditions.
2. Apply strict URL validation before calling the `curl` command.
3. Validate that the `functions` fetched are expected and don't contain malicious code before using `eval`.
4. Add more error handling for operations like creation of directories and modifying file permissions.
5. Avoid logging sensitive data for better security.

2.0.576 `hps_log`

Contained in `lib/functions-core-lib.sh`

Function signature: e571617bb441a3935bc9ef014800d6243fb9dc0df427aa2d194d8715c690ab8f

2.0.577 Function overview

The `hps_log` function is a logging utility for Bash scripts that maps the HPS log levels to syslog priorities and writes them into a log file. It uses HPS (High-Performance Systems) log levels of INFO, WARN, ERROR and DEBUG and if an unrecognized level is provided, it defaults to INFO. The function also allows to set an identity and log directory via environment variables. The log entry includes a timestamp of the format '%Y-%m-%d %H:%M:%S'.

2.0.578 Technical description

- **name:** `hps_log`
- **description:** This function is a logging utility that maps HPS log levels to syslog priorities. It writes log with a timestamp to a specified log file.
- **globals:**
  - `HPS_LOG_IDENT`: Describes the identity for the log entries. Defaults to 'hps'.
  - `HPS_LOG_DIR`: The directory where the log files are stored.
- **arguments:**
  - `$1`: Log level. Possible values: ERROR, WARN, INFO, DEBUG. Default is INFO.
  - `$*`: The log message.

- **outputs:** Log line containing timestamp, identity, log level and log message written into `hps-system.log` in the specified HPS_LOG_DIR.
- **returns:** N/A
- **example usage:**

```
HPS_LOG_DIR="/path/to/logs" hps_log "ERROR" "Critical failure in
 ↪  module ABC"
```

2.0.579 Quality and security recommendations

1. Use better input validation for the log level to avoid undesired results when incorrect inputs are provided.
2. Create a more robust way of handling missing or erroneous input parameters to the function.
3. Implement fail-safe measures to ensure that the logging operation does not crash the script in case it fails.
4. Protect the log file and ensure it has the correct permissions to avoid unauthorized access or manipulation.
5. For secure or sensitive environments, consider adding encryption to log files or entries to protect sensitive data from being exposed.

2.0.580 `hps_origin_tag`

Contained in `lib/functions.d/system-functions.sh`

Function signature: 34e636eb4b7c0c9bf98b49ce8416227a48485bd990954c5b7f74feba9f1c472a

2.0.581 Function Overview

The `hps_origin_tag` function attempts to generate a unique tag based on the origin of a given process. The function considers several aspects such as an override option, user, host and process ID in the context of an interactive terminal, and also client IP/MAC in the context of a non-interactive terminal.

2.0.582 Technical Description

- **Name:** `hps_origin_tag`
- **Description:** This function generates a unique tag indicating the origin of a process. It first checks if an explicit override is provided. If the script is running from an interactive terminal, the function captures the user, host, and process ID. If the script is running from a non-tty environment (e.g., a web server), it attempts to use client IP/MAC information to generate the tag.
- **Globals:** `REMOTE_ADDR: Internet protocol address of remote computer`

- **Arguments:** `$1: Overrides the need for automatic origin determination`
- **Outputs:** Prints a string that stands as the unique origin tag. The format could be process ID, user-host data, IP or MAC address.
- **Returns:** `0` on successful execution
- **Example Usage:** `tag=$(hps_origin_tag)`

2.0.583 Quality and Security Recommendations

1. Ensure proper validation and sanitation of command outputs like `id -un` and `hostname -s` to prevent any potential command injection attacks.
2. Use stringent error handling and check the return codes of executed commands as much as possible.
3. Avoid putting sensitive data like MAC addresses within origin tags as they can leak data by exposing it in logs or other output. If it is necessary, make sure logs/output storing these tags are adequately secured.
4. When printing out the tag, consider using an appropriate log level.
5. If the function fails to create a tag, it would be advisable to include fallback methods or return a standard error code.

2.0.584 `hps_reload`

Contained in `lib/functions.d/node-bootstrap-functions.sh`

Function signature: 189ec6ad5a00e5ecfa2db002ef7e0c7a9c7d69e9bf0f3ab81ef1f0a318e5d108

2.0.585 Function Overview

The `hps_reload` function is a bash script used to reload Node function configurations. Essentially, it calls another function, named `hps_load_node_functions`, which updates or loads Node.js functions into the current environment.

2.0.586 Technical Description

```
- name: hps_reload
- description: A bash script function that reloads Node.js function
 ↪  configurations by calling hps_load_node_functions function.
- globals: [ hps_load_node_functions: the function being called ]
- arguments: [ This function does not take any arguements ]
- outputs: The output of this function would be entirely dependent
 ↪  on what the hps_load_node_functions does. The hps_reload
 ↪  itself doesn't produce a visible output.
- returns: --
- example usage: hps_reload
```

2.0.587 Quality and Security Recommendations

1. Always ensure that the `hps_load_node_functions` is safely declared before this function is called. An error may occur if the `hps_reload` function tries to call an undefined function.
2. Ensure that proper permissions are set for the function and the script containing this function, to maintain system integrity and script function.
3. Use clear naming conventions for function names to prevent confusion and maintain code clarity.
4. Frequent confirmation and testing of function utility is recommended, to check if it performs as required.
5. Incorporate additional security measures such as input validation if arguments are expected to be passed in future updates of the function, to avoid code injection attacks.
6. Comprehensive error checking and handling mechanisms are also recommended, to improve overall robustness of the function.

2.0.588 `hps_safe_eval`

Contained in `lib/functions-core-lib.sh`

Function signature: 73eb16364c068429e1a374db940c3e966cd153931b38db0d3d8181f676190ae0

2.0.589 Function Overview

The function `hps_safe_eval` is designed for secure execution of shell commands in Bash. It accepts a block of shell commands as the first argument, and an optional description of that command as the second argument. The function attempts to execute the passed commands and obfuscates any error messages that may arise. If the command fails to execute, it will call a debugging function `hps_debug_function_load` and return an exit status of 1, otherwise it will return 0.

2.0.590 Technical Description

- **Name**: hps_safe_eval
- **Description**: A function to securely evaluate and execute a block of code in bash. If evaluation fails, it provides diagnostic information using `hps_debug_function_load`.
- **Globals**: None.
- **Arguments**:
  - $1: The block of shell commands (‘code’) to be securely evaluated.
  - $2: A description for the block of command (‘desc’). This is optional, and if not provided it will default to the string ‘code’.
- **Outputs**: If the passed block of commands cannot be securely evaluated, it will output error and diagnostic messages to the standard error (stderr).

- **Returns**: Returns 0 if the block of commands is successfully evaluated, otherwise returns 1.
- **Example usage**: `hps_safe_eval 'ls -l' 'List Files'`

2.0.591 Quality and Security Recommendations

1. Error messages should be more descriptive to give the user additional detail about why the evaluation failed. This could be achieved by including the output of `eval` in the error message.
2. `eval` should only be used for evaluation of trusted code to avoid command injection attacks. Check the origin and integrity of the block of commands before passing it to `hps_safe_eval`.
3. Avoid using eval where possible; consider safer alternatives such as `printf -v` or Bash arrays.
4. Ensure that all input variables that are passed to `hps_safe_eval` are sanitized to avoid shell injection and other potential security risks.
5. The function should accept multiple blocks of commands as input, rather than just one, to increase its flexibility and utility. This will require iterating over the input arguments.
6. Consider handling the error within the function itself rather than just returning an exit status. This may make it easier for users to handle errors in their scripts.
7. Regularly update and audit `hps_debug_function_load` to prevent the debug information from falling into the wrong hands.

2.0.592 `hps_services_restart`

Contained in `lib/functions.d/system-functions.sh`

Function signature: ecd142eb37678b0067ceb8a949072b8c63fbafcaead2d1a4a19e660b417d7871

2.0.593 Function Overview

The `hps_services_restart` function handles the restart of all processes managed by Supervisord in a specific directory. First, it executes the `_supervisor_pre_start` function and then restarts all the supervisor processes. The cluster service directory path is fetched by the `get_path_cluster_services_dir` function.

2.0.594 Technical Description

- **Name**: `hps_services_restart`
- **Description**: This function is designed to handle the restart of all Supervisord managed processes. It first calls the `_supervisor_pre_start` function to ensure pre-start conditions are met. Then, it uses `supervisorctl` with the

restart command to restart all services in the cluster services directory. The path to this directory is obtained using `get_path_cluster_services_dir`.

- **Globals**: None
- **Arguments**: None
- **Outputs**: Outputs the status of the restart command.
- **Returns**: None
- **Example usage**: To restart all the services, simply call `hps_services_restart` in the bash shell.

2.0.595 Quality and Security Recommendations

1. Regularly update Supervisord to patch known security vulnerabilities.
2. Handle logging effectively to trace any potential issues that may occur during service restart.
3. Check the return status of the `_supervisor_pre_start` function and handle any pre-start conditions that aren't met.
4. Validate outputs from the command run using `supervisorctl` to handle any exceptions and restart failures.
5. Enforce necessary security privileges to ensure that the function cannot be misused for any unintended purposes.
6. Include error handling measures to help in debugging and to ensure smooth execution.

2.0.596 `hps_services_start`

Contained in `lib/functions.d/system-functions.sh`

Function signature: 5dfbfd6dea999f8dae840b8f27d4a95752d251d6b56a27239df23831d864dfbf

2.0.597 Function Overview

The function `hps_services_start` is used to initiate all services that are managed by Supervisor. The function first assures that the Supervisor process is ready for the initiation of its services by calling the function `_supervisor_pre_start`, then logs the start of all services under the management of Supervisor. The desired supervisor configuration file is set by the function `get_path_cluster_services_dir`.

2.0.598 Technical Description

- Name: `hps_services_start`.
- Description: The function starts all services managed by Supervisor. It calls the `_supervisor_pre_start` function first, making all preparations before starting the services. A log is provided displaying the status of the services starting up.

- Globals: None.
- Arguments: None.
- Outputs: The function outputs an information log entry for the starting of all services through the `hps_log` function.
- Returns: The function does not explicitly return a value.
- Example Usage:

```
hps_services_start
```

2.0.599 Quality and Security Recommendations

1. Include error handling: Currently, the function assumes that `_supervisor_pre_start` and `hps_log` will always succeed. Adding error handling for these function calls can help the function react correctly to any issues that might occur.
2. Set permissions properly: Carefully consider the required permissions for the scripts using this function. Avoid overly permissive settings that could be exploited by an attacker.
3. Logging: "info" level logging is currently used, but depending on its deployment context, it might be worth considering logging at the "debug" level for more detailed logs, though at the cost of more storage space.
4. Include input validation: Presently, the function does not take in any arguments. Should this change in the future, ensure that the function includes validation to prevent potential security issues such as command injection.

2.0.600 `hps_services_stop`

Contained in `lib/functions.d/system-functions.sh`

Function signature: 8389cadf05a55bb98c0cc0b40a84671a6cc4e156fdaaafee6f327e84dff29f00

2.0.601 Function overview

The function `hps_services_stop()` is utilized to halt all running services under the control of the Supervisor program. The function operates by calling the supervisor control command (`supervisorctl`) and specifying the path to the Supervisor configuration file, which obtains from invoking another function (`get_path_cluster_services_dir`). It's an essential function for maintaining system stability and clean shutdown processes.

2.0.602 Technical description

- **Name:** `hps_services_stop()`
- **Description:** This function stops all services managed by Supervisor, by calling the `supervisorctl` command along with the path to the configuration file.
- **Globals:** None.

- **Arguments:** None.
- **Outputs:** The function outputs any notifications or errors from the `supervisorctl` command. If successful, it will output standard messages from `supervisorctl` signifying the successful stopping of all services.
- **Returns:** It might return exit codes from the `supervisorctl` command. In general, if all services stopped successfully, a zero (signifying success) will be returned. Non-zero if any error occurs.
- **Example usage:** `hps_services_stop` (since the function does not require arguments, it can be invoked directly via the function name)

2.0.603 Quality and security recommendations

1. Ensure proper user permissions: Make certain the function is only executed by users with proper permissions to halt the services. Access to such commands should be strictly regulated.
2. Error Handling: Incorporate error handling to catch any issues that might arise from the `supervisorctl` command. It may not always be able to halt services for various reasons.
3. Logging: Implement logging to keep track of when and why specific services were stopped. This can be beneficial for auditing and problem-solving purposes.
4. Documentation: Maintain clear documentation of this function to facilitate its use and maintenance.
5. Secure Path: Avoid manipulating the function such that the path input for the configuration file could be substituted or tampered with by an untrusted source.

2.0.604 `hps_source_with_debug`

Contained in `lib/functions-core-lib.sh`

Function signature: be480631c241282ddd867118c1ded55f7580182f64222610f58bc00abaffaea6

2.0.605 Function overview

The function `hps_source_with_debug()` is a helpful function that you can use in Bash scripting. It's primary function is to read and execute commands from the file specified in the first argument, suppressing error output. If the sourcing fails, it will print error message, optionally load debug function and if specified, check the syntax of the file using bash or continue the execution regardless of the errors.

2.0.606 Technical description

name:

`hps_source_with_debug()`

description:

This function reads and executes commands from a file whose path is given by the first argument. It suppresses any error output originating from this operation. If reading from the file fails, the function prints a customizable error message and checks the syntax of the file. It also has an option to continue the execution even if there are errors.

globals:

None

arguments:

- `$1`: The path to the file to read from.
- `$2`: (Optional) If set to "continue", the function will continue execution despite encountering errors.

outputs:

The function will output error messages and the results of the syntax check, if there are any errors while reading from the file.

returns:

The function will return 0 if the source command was successful, otherwise it will return 1.

example usage:

```
hps_source_with_debug "/path/to/somefile.sh"
hps_source_with_debug "/path/to/somefile.sh" "continue"
```

2.0.607 Quality and security recommendations

1. To make the function more robust, it should check whether `$1` is empty or whether the file at the path specified by `$1` exists and is readable before attempting to source it.
2. For better error handling, consider adding a `set -e` at the beginning of the function and `set +e` at the end.
3. For security reasons, be wary of sourcing a file that may contain malicious or incorrect code. This function should only be used with trusted files.
4. Ensure that the path to the file (`$1`) isn't coming from an untrusted source or user input to prevent potential command injection issues. Consider using a static code analysis tool to further enhance security.

2.0.608 `hps_status`

Contained in `lib/functions.d/node-bootstrap-functions.sh`

Function signature: a980623350909b8aab94458b30ca951c620a783779e027dddad917fc7ec7ebcb

2.0.609 Function Overview

This function, `hps_status()`, displays the status of High-Performance Storage (HPS) Bootstrap Library. It prints the Provisioning node, Distribution, Library version, and whether the Node functions are loaded or not. The functionality is achieved through `echo` and specific command calls.

2.0.610 Technical Description

- **Name:** `hps_status()`
- **Description:** This is a function defined in Bash. It is used to display the status of the HPS Bootstrap Library.
- **Globals:** N/A
- **Arguments:** N/A
- **Outputs:** Custom console messages regarding the status of the HPS Bootstrap library and Node functions.
- **Returns:** It does not return anything because it just prints on the console.
- **Example usage:** Simply call the function using `hps_status`.

```
hps_status
```

2.0.611 Quality and Security Recommendations

1. The function would be safer if it explicitly managed potential errors that may occur while trying to retrieve the provisioning node and distribution string.
2. Currently, there is no input verification. Although the function does not use any arguments, it is good practice to always verify the input.
3. The function could benefit from added comments to describe what each part of the function does. Although it is simple and readable as-is, as the complexity increases, comments can make it easier to understand.
4. The function depends heavily on the existence of other functions (e.g., `hps_get_provisioning_node` and `hps_get_distro_string`). Ensure these components are robust and secure.
5. While security is less of a concern because no sensitive data appears to be handled, be mindful of any changes that may introduce such data handling.
6. The function could potentially use more robust error handling when checking if node functions are loaded or not.

2.0.612 `hps_url_encode`

Contained in `lib/functions.d/node-bootstrap-functions.sh`

Function signature: af2671ff9ba09a5ac5edf1094446f99b2184d853e9ac997683b95f8bd7f0e996

2.0.613 Function overview

The function `hps_url_encode` is a Bash function that encodes a given string into URL encoding format. The function takes in one argument (a string) and iterates through each character of the string. If the character falls within the predefined set `[a-zA-Z0-9.~_-]`, it remains the same. However, if it falls outside of this set, it is converted into its corresponding hexadecimal equivalent.

2.0.614 Technical description

- **Name**: `hps_url_encode`
- **Description**: This bash function encodes a string into URL encoding format. It retains alphanumeric characters and a few special characters as they are and converts all other characters into their hexadecimal equivalents preceded by a '%'.
- **Globals**: None
- **Arguments**:
  - `$1: s` This is the input string for the function to encode.
- **Outputs**: Prints the URL encoded form of the input string to stdout
- **Returns**: None
- **Example usage**: `bash hps_url_encode "Hello World!"` This will output "Hello%20World!".

2.0.615 Quality and security recommendations

1. Check for empty string: Add an initial condition to check if the input string is empty. If it is, the function should return an appropriate error message.
2. Usage of `printf`: The function uses `printf` to print the encoded string, this is potentially unsafe due to `printf`'s ability to evaluate variables as format strings. Always prefer echo or ensure correct usage of `printf`.
3. Error Handling: Add error handling logic to deal with any unforeseen input or edge cases, such as binary data.
4. Testing: Include test cases to verify the functionality with different types of input strings – like string with spaces, special characters etc.
5. Documentation: Always keep function documentation up to date. This not only benefits other developers but also makes security audits easier.

2.0.616 `init_dns_hosts_file`

Contained in `lib/functions.d/dns-dhcp-functions.sh`

Function signature: 9b28515652d3fd4d01125906b949fc2a95fa44ddced3687030f35955b86730a0

2.0.617 Function Overview

The function `init_dns_hosts_file` is used to initialize a DNS hosts file. It does this by setting parameters and creating necessary directories and files. Further, it extracts necessary variables like DNS_DOMAIN and DHCP_IP from the existing configured system, validates them and then adds an IPS entry into the DNS hosts file. The function logs errors and success messages at different stages in the process.

2.0.618 Technical Description

- **Name**: `init_dns_hosts_file`
- **Description**: Initializes a DNS hosts file by creating necessary directories, files, extracting variables from the configuration, validating variables and adding an IPS entry into the DNS hosts file.
- **Globals**:
  - *None*
- **Arguments**:
  - *None*
- **Output**: Creates a DNS hosts file with necessary entries. Also prints out info and error log messages.
- **Returns**: The function returns 1 in case of any error. If executed successfully, it returns 0.
- **Example Usage**: The function is invoked without any arguments like so: `init_dns_hosts_file`

2.0.619 Quality and Security Recommendations

1. The function relies heavily on other functions like `get_path_cluster_services_dir`, `hps_log`, `cluster_config get DNS_DOMAIN`, `strip_quotes`, etc. Any changes or errors in these functions could also impact this function.
2. Validate and sanitize data received from external resources. For instance, some data is fetched from the `cluster_config` and `DNS_DOMAIN` which if corrupted, may cause issues.
3. Write comprehensive error messages that reflect precise failure points. This can aid debugging.
4. If possible, consider error handling where the directories or files couldn't be created instead of simply logging error and returning 1. One possible scenario includes permissions issues.
5. Occasionally, the file may not be able to get cleaned up in case of errors resulting in `touch "$dns_hosts_file"`. Ensure appropriate file cleanup actions after error logging.

2.0.620 `initialise_cluster`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: d5981ec28473618b5cd51e9c3ade1ff3471cc55b1c1da368f886f31feec27ee4

2.0.621 Function Overview

The `initialise_cluster` function is a Bash function designed to setup and initialize a new cluster. The function checks if the user has provided a cluster name and if the cluster directory already exists. If the cluster name is not provided or if the cluster directory already exists, it produces an error log and terminates. If the checks are successful, it will create a new directory structure for the new cluster along with a cluster configuration file. Finally, it will set the cluster's name in the configuration and call the function `export_dynamic_paths`. If this function call fails, it will log an error and terminate.

2.0.622 Technical Description

- **Name**: `initialise_cluster`
- **Description**: Initialises a new cluster by creating the necessary directory structure and cluster configuration file. Sets the cluster's name in the configuration. Calls `export_dynamic_paths` and exits if this function call fails.
- **Globals**: [`HPS_CLUSTER_CONFIG_BASE_DIR`: The base directory where cluster directories are located.]
- **Arguments**: [`$1` (cluster_name): User provided name for the new cluster]
- **Outputs**: Logs and errors to stdout
- **Returns**:
  - 0 if the function completes successfully
  - 1 if no cluster name is provided
  - 2 if the cluster directory already exists
  - 3 if the call to `export_dynamic_paths` fails
- **Example Usage**: Initialise a new cluster named "my_cluster" with `initialise_cluster "my_cluster"`

2.0.623 Quality and Security Recommendations

1. Make sure the variable `HPS_CLUSTER_CONFIG_BASE_DIR` is defined in a secure location that the current user has read, write, and execute permissions on.
2. Add input validation for the cluster name to prevent any possible command injection vulnerabilities.
3. User should have appropriate permissions to execute this function and handle the involved directory and files.
4. Function should handle exceptions and edge cases more efficiently.

5. Logging mechanism should be more specific about the types of errors and conditions occurring during the function execution.

2.0.624 `_init_matches_metadata`

Contained in `lib/functions.d/node-libraries-init.sh`

Function signature: f87c1bf49a6c8d20d71073d2222dbec45163da2cf512c39001310bdd198bc874

2.0.625 Function Overview

This Bash function, `_init_matches_metadata()`, is designed to handle metadata in initialization files. The function defines and reads different types of metadata, checks for specific flags such as `RESCUE=true`, and matches each metadata field.

2.0.626 Technical Description

- **Name:** `_init_matches_metadata`
- **Description:** This script is intended to initialize different types of metadata. It processes an initialization file and some input parameters (check parameters) to determine if the metadata matches the check parameters. It also takes into account a RESCUE flag.
- **Globals:** None
- **Arguments:**
  - `$1`: `init_file` - The file from which to read the first line.
  - `$2`: `check_os` - Operating system type to check against metadata.
  - `$3`: `check_type` - Type of metadata to check.
  - `$4`: `check_profile` - Profile information to check against metadata.
  - `$5`: `check_state` - State information to check against metadata.
  - `$6`: `check_rescue` - If the function should check for a RESCUE flag (default = false).
- **Outputs:** Logs a debug message if debug mode is activated.
- **Returns:** `0` if metadata matches the check parameters or if there's no metadata; `1` if checkings fail or if a RESCUE init is being executed without the RESCUE flag being set to "true".
- **Example Usage:** `bash _init_matches_metadata "./myfile.txt" "linux" "type1" "profile1" "state1" "true"`

2.0.627 Quality and Security Recommendations

1. Strongly consider checking the existence and validity of the initialization file (`init_file`) before execution. This function can fail in cases where the file does not exist or cannot be read.

2. Make sure that the `head` command execution does not error when trying to read the file, or in case of an empty file. Overlooking the error stream `2>/dev/null` might hide potential issues.
3. Enhance the metadata extraction part by handling unexpected formats more robustly, so the function does not crash or behave unexpectedly when faced with malformed metadata.
4. When running in debug mode, sanitize the output to avoid leaking sensitive information contained in the metadata.
5. Aim for more precision in the match field function (`_matches_field`). Currently, it returns 1 if no match is found without discerning whether it's due to an absence of data or a mismatch.

2.0.628 `install_opensvc_foreground_wrapper`

Contained in `lib/functions.d/opensvc-functions.sh`

Function signature: 044d09f5864508cf501804626eabf1e2282f7b5c3a9c290f490b470dcca0b251

2.0.629 Function Overview

The function `install_opensvc_foreground_wrapper` is used to install a bash wrapper for the OpenSVC agent to run in the foreground. Its output is directed to a log directory, either specified by the user or defaulted to `/srv/hps-system/log`. If the required directories don't exist, the function will create them. This function performs a preflight check to ensure an agent key exists and is not empty. If the agent key is missing or empty, the function echoes an error message and exits with a status of 2. When the content of the bash wrapper is prepared, it then checks if the target file exists and only replaces the target if the contents therein differ from the intended set of contents.

2.0.630 Technical Description

- **Name**: install_opensvc_foreground_wrapper()
- **Description**: A Bash function that safely installs an OpenSVC agent bash wrapper script for running the agent in the foreground. Handles pre-flight checks and facilitates logging output of the agent.
- **Globals**: [HPS_LOG_DIR: User defined log directory or defaulted to `/srv/hps-system/log`]
- **Arguments**: None.
- **Outputs**: Writes a bash wrapper script with specific contents at a target location. It sends output to either user defined `HPS_LOG_DIR` or default path `/srv/hps-system/log`.
- **Returns**: Returns 1 if temporary file creation fails during execution. Returns 0 if the function completes successfully.
- **Example Usage**: Install OpenSVC wrapper in the foreground: `install_opensvc_foreground_wrapper`

2.0.631 Quality and Security Recommendations

1. To prevent file path injection issues, it is recommended to further validate the HPS_LOG_DIR global variable value before use.
2. The preflight check on the presence and emptiness of /etc/opensvc/agent.key is a good practice. You can enhance it by additionally checking the validity and correct format of this key file.
3. To prevent a full disk from causing a complete system halt, implement disk usage checks and automatic clean-ups of old log files in the log generation process. Avoid writing logs directly to critical locations like /var/log.
4. Consider adding shell option `set -u` at the start of the functions to prevent the script from running with unset variables, as this can cause unexpected behavior.
5. Utilize more detailed logging methods to capture more comprehensive logs for complex error scenarios. Log outputs of critical operations can help with debugging later on.

2.0.632 int_to_ip

Contained in lib/functions.d/network-functions.sh

Function signature: 14a6ccb9401350f58647bcc5d3c386b17ebd884462480f7e5ed07df11d092d11

2.0.633 Function Overview

The Bash function int_to_ip is designed to convert an integer to its corresponding 32-bit IPv4 address format (e.g., 192.168.0.1). It accepts an integer as an argument, performs bit shift operations and bitwise AND with 255 on this integer in order to separate out four octets that make up an IPv4 address, and then echoes the result in the format of a standard dotted-notation IP address.

2.0.634 Technical Description

- **Name**: int_to_ip
- **Description**: Converts a given integer value to its equivalent IP address in 32-bit IPv4 format.
- **Globals**: None
- **Arguments**:
  - $1: The input integer to be converted to IPv4 address format.
- **Outputs**:
  - The calculated IP address in 32-bit IPv4 format is echoed to stdout.
- **Returns**:
  - The function will always return 0 to indicate successful execution. Errors are not covered in this function.
- **Example Usage**:

- `int_to_ip 3232235776` will echo `192.168.1.0` to stdout.

2.0.635 Quality and Security Recommendations

1. **Error Handling**: As a good practice, this function should also handle error situations such as incorrect input types and out-of-range input values. These checks can be added at the beginning of the function.
2. **Input Validation**: Validate the input to ensure that it is a positive numeric value and falls within the valid range for a 32bit IP address.
3. **Return Codes**: Make use of different return codes to indicate different kinds of issues (e.g., invalid input, error while converting), this would help the calling function understand if there were any issues during execution.
4. **Commenting**: Include more comments throughout the function to ensure maintainability and comprehension for other developers.
5. **Consistent Coding Style**: Make sure there is consistency in the use of quotations and other coding elements within the function, following the best practices and guidelines for Bash scripting. This will help in maintaining the readability of the code.
6. **Security**: Consider the security implications, always sanitize and validate the input parameters to prevent potential code injection attacks.

2.0.636 `ips_allocate_storage_ip`

Contained in `lib/functions.d/network-functions.sh`

Function signature: 7dcadc01ce4c2633c148412e7ddeaa6b83b7f1be5339d9afcd614f25a2171b25

2.0.637 Function Overview

The `ips_allocate_storage_ip` function is designed for allocating storage IP addresses in a given network storage environment. It takes two arguments: `storage_index` and `source_mac`. The function validates these arguments and also checks if the storage network is properly configured. If all checks are successful, it calculates the VLAN ID and validates the configuration. If an IP allocation already exists for the source_mac, it returns the existing allocation, otherwise, it looks for the next available IP to allocate.

2.0.638 Technical Description

- **Name**: `ips_allocate_storage_ip`
- **Description**: Allocates storage IP addresses in a storage environment where multiple VLANs are used for traffic segregation.
- **Globals**: None.
- **Arguments**:

  - `$1 (storage_index)`: Index of the storage to allocate IP address.
  - `$2 (source_mac)`: The MAC address of the source requesting the IP allocation.
- **Outputs**: Prints VLAN, IP, Netmask, Gateway, MTU details if successful or error messages during failures.
- **Returns**: Returns 0 if a new IP address was successfully allocated or an existing allocation was found; returns 1 in case of errors such as missing or invalid arguments, uninitialized storage network, unavailable IPs, failed subnet parsing, etc.
- **Example usage**: `ips_allocate_storage_ip 0 "00:11:22:33:44:55"`

2.0.639 Quality and Security Recommendations

1. There is a chance of IP address exhaustion since the function starts searching for available IP addresses at `.100` and stops at `.250`. Consider implementing a more robust method for allocating IP addresses to prevent exhaustion, e.g., using DHCP.
2. Consider setting up detailed logging at every step, so that errors and issues can be tracked down more easily.
3. Implement input sanitization to prevent potential security issues resulting from unexpected or malicious input.
4. Proper error handling steps should be followed to ensure that the function fails gracefully in case of errors.
5. Implement helper function testing to ensure consistency and reliability of helper functions used in the main function.
6. The allocation process uses recursion which could lead to stack overflows if not handled properly. Ensure that potential recursive errors are anticipated and handled.

2.0.640 `ips_install_opensvc`

Contained in `lib/functions.d/opensvc-function-helpers.sh.sh`

Function signature: 93c2b6318f19afe368284bf486b55dc8a2a89b582264cabd4713b297b7c7a749

2.0.641 Function Overview

The `ips_install_opensvc` function is a part of Bash scripting that is designed to ensure presence, install, fix dependencies and sanity-check 'om'. The function first checks if the .deb package of opensvc is available and non-empty in the `$HPS_PACKAGES_DIR/opensvc/` path. If the package is missing or empty, the function returns an error. Next, the function updates the apt-get, installs the opensvc, and removes any unessential files from the Debian libraries were installed earlier. Finally, the function sanity checks if the 'om' command is available and runs it to print a version. If the 'om' command cannot be found, the script will return an error.

2.0.642 Technical Description

- **Name**: `ips_install_opensvc`
- **Description**: This function verifies the presence of an opensvc package, installs it, resolves any missing dependencies from Debian repositories and performs a sanity check on the command 'om'.
- **Globals**: `$HPS_PACKAGES_DIR`: The directory where the opensvc .deb package is located. `$OSVC_DEB`: The opensvc .deb package.
- **Arguments**: None.
- **Outputs**: Debug, error logs to stdout.
- **Returns**: 1 if the .deb package is missing, empty, or if the 'om' command is not found. Otherwise, the function successfully installs the package and no value is returned.
- **Example usage**: `ips_install_opensvc`

2.0.643 Quality and Security Recommendations

1. Add input validation checks: Ensure that the variables used within the function are properly defined before the function is performed.
2. Implement error handling: Make sure to handle the scenarios where the apt-get commands fail.
3. Use HTTPS for package download: If the packages are being downloaded from a remote server, ensure it is done over HTTPS to protect against MITM (Man-In-The-Middle) attacks.
4. Check for command injection: Verify that there is no arbitrary command injection possible via the global variables.
5. Regularly update the system and packages: To protect against known vulnerabilities, always keep both the system and installed packages up to date.
6. Provide user feedback: If the function successfully completes its operation, it should return a success message instead of no value returned. This would improve user experience and debugging.

2.0.644 `ips_install_opensvc`

Contained in `lib/functions.d/opensvc-function-helpers.sh.sh`

Function signature: 93c2b6318f19afe368284bf486b55dc8a2a89b582264cabd4713b297b7c7a749

2.0.645 Function Overview

The function `ips_install_opensvc` is used for verifying the presence, installation, fixing dependencies, and sanity-checking the 'om' (OpenSVC Manager) command. This function particularly checks if a `.deb` package for OpenSVC exists and is non-empty in a predefined directory. If the `.deb` package is found, it attempts to install it while resolving

any missing dependencies from the Debian repositories. Finally, the function validates successful installation by checking the availability of the 'om' command and ensuring it can print a version.

2.0.646 Technical Description

- **Name**: `ips_install_opensvc`
- **Description**: This function verifies the existence of a valid `.deb` package for OpenSVC, installs it, fixes any missing dependencies from Debian repositories, and performs a sanity check to ensure the successful installation of the 'om' command.
- **Globals**: [OSVC_DEB: A variable that holds the most recent `.deb` file from the OpenSVC directory]
- **Arguments**:
  - `$HPS_PACKAGES_DIR`: a directory path for packages
- **Outputs**: Logs for debugging and error information.
- **Returns**: `1` if the `.deb` package is missing, empty, or if the 'om' command is not found after installation; otherwise doesn't return a value.
- **Example usage**:

```
HPS_PACKAGES_DIR=/path/to/packages
ips_install_opensvc
```

2.0.647 Quality and Security Recommendations

1. Validate and sanitize the user input `$HPS_PACKAGES_DIR` to prevent potential directory traversal issues.
2. To improve reusability, consider passing the package name as an argument rather than being hardcoded.
3. Catch and appropriately sanitize or handle errors related to `apt-get` commands to prevent any potential security issues or failures.
4. Use more secure alternatives than `rm -rf /var/lib/apt/lists/*` which could lead to unintentional deletion of important data.
5. Consider adding more descriptive and detailed logs, especially for errors, to make it easier for debugging and solving potential issues.

2.0.648 `_ips_resolv_conf_update`

Contained in `lib/functions.d/dns-dhcp-functions.sh`

Function signature: 360453a9bdc29599bd8d559a36b42bdb89b837fb625135b094a7b14b1b859daf

2.0.649 Function Overview

This function, `_ips_resolv_conf_update()`, is designed to update the `/etc/resolv.conf` file, which is typically used to configure DNS servers on

a Unix-based operating system. It retrieves the IPS address and DNS domain through predefined functions `get_ips_address` and `cluster_config get DNS_DOMAIN` respectively. It then writes these into `/etc/resolv.conf` in the correct format.

2.0.650 Technical Description

- **Name:** `_ips_resolv_conf_update`
- **Description:** This function updates the `/etc/resolv.conf` file with nameserver and search details corresponding to `ips_ip` and `dns_domain`.
- **Globals:** [`ips_ip`: IPS address, `dns_domain`: Domain name for DNS]
- **Arguments:** There are no arguments passed to this function
- **Outputs:** A new `/etc/resolv.conf` file is created and written to with nameserver (`ips_ip`) and search (`dns_domain`) parameters.
- **Returns:** The function will return `1` (indicating failure) in two scenarios, if either IPS address is not available or DNS_DOMAIN configuration is not successful. Otherwise, it will return `0` (indicating success).
- **Example usage:** `_ips_resolv_conf_update`

2.0.651 Quality and Security Recommendations

1. Add error checking to ensure that the function only works if the execution user has the necessary permissions, particularly write access, to the `/etc/resolv.conf` file.
2. Add test conditions to verify the validity of the values for `ips_ip` and `dns_domain`.
3. Include logging for potential errors and successful completions for better debug capabilities. Currently, the logging only writes when errors occur.
4. Add a check to see if `get_ips_address` and `cluster_config get DNS_DOMAIN` functions exist to avoid any unexpected function not found errors.
5. To improve security, consider limiting the exposure of the `resolv.conf` file while it is being modified.

2.0.652 `_ips_resolv_conf_update`

Contained in `lib/functions.d/dns-dhcp-functions.sh`

Function signature: 360453a9bdc29599bd8d559a36b42bdb89b837fb625135b094a7b14b1b859daf

2.0.653 Function Overview

The `_ips_resolv_conf_update` function is a Bash function designed to update the resolv.conf file, which is used by the resolver library to resolve hostnames into IP addresses. It fetches the IP address using `get_ips_address` function and DNS

domain from the cluster's configuration. If either of these steps fails, it logs an error message and exits with a non-zero status code. If these operations are successful, the function updates `/etc/resolv.conf` with the new nameserver and search domain, logs an info message, and returns 0.

2.0.654 Technical Description

- **Name**: `_ips_resolv_conf_update`
- **Description**: A bash function designed to update the `/etc/resolv.conf` file with IP address generated by `get_ips_address` and DNS domain from clusters configuration.
- **Globals**: None
- **Arguments**: None
- **Outputs**: If executed successfully, a new `/etc/resolv.conf` file is created. Messages to the standard output in case of errors.
- **Returns**:
  - `1`, in case of any errors while fetching IP address or DNS domain.
  - `0`, after successfully updating the `/etc/resolv.conf` file.
- **Example Usage**:

  ```
  _ips_resolv_conf_update
  ```

2.0.655 Quality and Security Recommendations

1. Incorporate validation checks to confirm that the returned IPS address and DNS domain are in the expected formats.
2. Consider adding more detailed logging. Include, for instance, the actual error message that caused the failure.
3. Avoid writing directly to `/etc/resolv.conf`. Preferably, write to a temporary file, then move that temporary file to `/etc/resolv.conf`.
4. Always perform file operations securely and in a robust manner.
5. Consider using more clear and descriptive function and variable names to improve the readability and maintainability of the code.

2.0.656 `ip_to_int`

Contained in `lib/functions.d/network-functions.sh`

Function signature: e1516fc19276d7592ec5f66f1d05a17ea74eeb43a3c726a9bb01fa86aa2a0b9b

2.0.657 Function Overview

This function named `ip_to_int`, takes an IP address as an argument and converts it into its equivalent integer representation. This is a common need in network programming. The function first validates the input using the `validate_ip_address` function. If the validation fails, it logs an error message using the `hps_log` function with the level set to error and then prematurely returns from the function with a return code of 1. If the validation succeeds, the IP address is then segmented into its four octets using the Internal Field Separator (IFS) and read into variables. Each octet is then bit-shifted appropriate number of places to the left, resulting in the equivalent integer value of the IP address.

2.0.658 Technical Description

- **Name:** `ip_to_int`
- **Description:** This function converts an IP address in string format to its equivalent integer representation.
- **Globals:** None
- **Arguments:**
  - `$1: IP address in string format.`
- **Outputs:**
  - If successful, prints the integer representation of the IP address to STDOUT.
  - If not, logs an error message using the `hps_log` function.
- **Returns:**
  - 0 if the IP address was successfully converted to integer.
  - 1 if the IP address is not valid.
- **Example usage:** `ip_to_int "192.168.1.1"`

2.0.659 Quality and Security Recommendations

1. The validation function `validate_ip_address` called by `ip_to_int` should have rigorous checks to ensure that only a valid IP address can pass through to help maintain the security of the systems involved.
2. It's recommended to refactor the `hps_log` outside of this function and handle errors in the caller function. This can improve the reusability of this function.
3. Utilize proper error handling mechanisms to ensure system stability in precipitous situations.
4. Incorporate comprehensive testing to ensure the function behaves as expected in various scenarios.
5. Always sanitize input data before it's used within the function to prevent injection attacks.

2.0.660 `ipxe_boot_alpine`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 7ee8212e40ee4318c7c978a9b2beb30c305edb63715fd058873baf009b4ae597

2.0.661 Function overview

This bash function, `ipxe_boot_alpine`, is designed to validate the Alpine repository and then boot using iPXE (Internet Packet eXchange Environment). iPXE is an open-source network boot firmware. The function gets the Alpine version for the MAC address, validates the Alpine repository, and then boots the operating system. If the Alpine repository validation fails, an error message is logged and the function returns. If the repository is confirmed to be valid, the function fetches configurations such as client IP and network CIDR, creates an apk overlay if it's missing, sets boot kernel arguments, and finally performs the boot operation via the `_do_pxe_boot` function.

2.0.662 Technical description

- **Name**: `ipxe_boot_alpine`
- **Description**: This function verifies the Alpine repository before attempting to boot the system. It further fetches OS configurations, generates an APK overlay, sets boot kernel arguments, and then proceeds to boot the system.
- **Globals**
  - `alpine_version`: The version of the Alpine system to be booted
  - `os_id`: The ID of the host Operating System
  - `client_ip`: The IP address of the client computer
  - `ips_address`: The IP address of the DHCP Server
  - `network_cidr`: The network ID in CIDR notation
  - `hostname`: Name of the host computer
  - `netmask`: The network mask
  - `download_base`: The base URL for the system to download the Alpine repository
- **Arguments**
  - `$mac`: The MAC address of the system to be booted
- **Outputs**: Outputs are all logged, including any error messages if the Alpine repository cannot be validated
- **Returns**: Returns 1 if the Alpine repository cannot be validated
- **Example usage**: `ipxe_boot_alpine "08:00:27:56:62:F1"`

2.0.663 Quality and security recommendations

1. Ensure user input is validated and sanitized to avoid common input-related flaws and injection attacks.

2. Enforce appropriate error handling so that the system does not fail or have unexpected behavior when it encounters an error.
3. Avoid using hardcoded IP addresses for DHCP and network related parameters. Instead, consider taking these IP addresses as arguments to the function or employ a mechanism to fetch them dynamically.
4. Consider logging all critical operations or errors for debugging and audit purposes.
5. Ensure the system is not logged in as a root user to avoid misuse or unauthorized changes.
6. It would be best if the operations within the function adhere to the principle of least privilege.

2.0.664 `ipxe_boot_from_disk`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: f26dc79cd8c9ea270e2758702b63d4a607fd3fd40737c5d947de898af37ff1fe

2.0.665 Function overview

The `ipxe_boot_from_disk` function is used to boot from a local disk via BIOS by exiting the iPXE stack. It first calls the `ipxe_header` function, prints a message to the console indicating that control is being handed back to BIOS for booting, sleeps for 5 seconds, and then exits.

2.0.666 Technical description

- **Name:** `ipxe_boot_from_disk`
- **Description:** This function facilitates booting from a local disk via BIOS by exiting the iPXE stack.
- **Globals:** None.
- **Arguments:** None.
- **Outputs:** Prints a message to the console indicating that control is being handed back to BIOS for booting.
- **Returns:** Nothing.
- **Example Usage:**

```
ipxe_boot_from_disk
```

2.0.667 Quality and Security Recommendations

1. As a good practice, it would be beneficial to add error handling or exceptions for certain operations such as checking if the BIOS processes are executing correctly when handling back control from iPXE.

2. It would also be useful to log the operations for debugging and auditing purposes.
3. From a security perspective, ensure that the appropriate permissions are set for the function to prevent unauthorized access or changes.
4. Avoid hardcoding values like sleep time. Instead, allow these to be configured through variables or configuration files.

2.0.668 `ipxe_boot_from_disk`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: f26dc79cd8c9ea270e2758702b63d4a607fd3fd40737c5d947de898af37ff1fe

2.0.669 Function overview

The `ipxe_boot_from_disk` function is designed to facilitate the process of booting from a local disk via BIOS by exiting the iPXE stack. The purpose of this function is to invoke the iPXE header and issue necessary commands to send control back to the BIOS, thus triggering it to boot from the local disk. A brief pause is also incorporated before exiting.

2.0.670 Technical description

- **name**: `ipxe_boot_from_disk`
- **description**: This function is used to boot from local disk via BIOS by exiting iPXE stack. It involves invoking iPXE header and issuing commands to hand control back to BIOS, along with a brief pause before completing its operation.
- **globals**: No global variables.
- **arguments**: No arguments.
- **outputs**: Outputs are the commands that are echoed- “Echo Handing back to BIOS to boot”, “sleep 5” and the exit command.
- **returns**: This function does not return any value.
- **example usage**: After defining the function, it can be invoked just by calling its name as follows:

```
ipxe_boot_from_disk
```

2.0.671 Quality and security recommendations

1. Error Handling: To improve upon this function, it is advisable to include error handling to account for any issues that might arise during the execution of the commands.
2. Logging: Adding logging statements could be beneficial for troubleshooting purposes.
3. Security: The function should have checks in place to ensure only authorized personnel can invoke a boot from the disk, reducing potential security risks.

4. Function Validation: Validate the outcome of each echo command to confirm it executed successfully.
5. Commenting: More comments can be included to clarify the role of each function step. This will make the code easier to evaluate and maintain.

2.0.672 `ipxe_boot_installer`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: f81efc5107ec0257f6dfff2fea55bf2aa9ba881078f51ad3de8d3661d72deca7

2.0.673 Function Overview

The function `ipxe_boot_installer()` is primarily used to configure network booting for a specified host by providing the MAC address. If the host type is not 'TCH', it sets the state to 'INSTALLING' and performs network boot via the function `ipxe_network_boot()`. If the host type is 'TCH', it sets up the host for network boot and applies the settings by rebooting.

2.0.674 Technical Description

- **name**: `ipxe_boot_installer()`
- **description**: This function is responsible for setting up the network boot for a particular host.
- **globals**: None.
- **arguments**:
  - `$1: mac` - This is the MAC address of the host.
- **outputs**: Logs about the operation status.
- **returns**: No value, but the function can potentially exit the script if the host type is not 'TCH'.
- **example usage**: `ipxe_boot_installer "00:11:22:33:44:55"`

2.0.675 Quality and Security Recommendations

1. Ensure that only valid MAC addresses are passed as arguments to the function.
2. Add more error checking and handling mechanisms to provide more resilient code.
3. Provide extensive logging for debugging purposes.
4. Sanitize and validate input data to prevent potential security risks.
5. Whenever possible, refrain from using `exit` in the function which can terminate the entire script. Use return statements instead.

2.0.676 `ipxe_boot_installer`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: f81efc5107ec0257f6dfff2fea55bf2aa9ba881078f51ad3de8d3661d72deca7

2.0.677 Function Overview

The `ipxe_boot_installer` function in Bash is designed to facilitate and configure a network boot installation for a host in a system network. It takes in the MAC address of the host and retrieves or sets various configuration details such as host type, host profile, system architecture and OS ID. It also defines the host's network configuration, sets the installation state, and initiatively kicks off the network boot if the host is not of type 'TCH'. For all others, it reboots the system to apply the network boot configuration.

2.0.678 Technical Description

- **Name**: `ipxe_boot_installer`
- **Description**: Given a MAC address, this function retrieves the host configuration details. Depending on the host type it sets corresponding state and triggers the network boot accordingly.
- **Globals**: [None]
- **Arguments**: [$1: MAC address of the host]
- **Outputs**: Messages indicating status of installation and potential error messages if the OS ID cannot be found.
- **Returns**: Does not explicitly return a value, the function carries out network boot installation.
- **Example Usage**: `ipxe_boot_installer "00:11:22:33:44:55"`

2.0.679 Quality and Security Recommendations

1. To ensure the quality of the function and prevent possible errors, it's recommended to validate the MAC address before processing.
2. Always handle user input (in this case, the MAC address) with care to prevent any potential security vulnerabilities such as injection attacks.
3. It might be beneficial to implement some form of error logging for better tracking and debugging purposes.
4. In cases where essential configuration values are not found or are invalid, the function should handle these issues gracefully - possibly with suitable fallbacks or error notifications.
5. To further improve security, consider using secure methods to handle sensitive data like the host configuration details.

2.0.680 `ipxe_cgi_fail`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 98eb961ae3ed7dfffc7f4ae68cd1c88aa5f47672ee53b71b29f0428922e8efaa

2.0.681 Function Overview

The function `ipxe_cgi_fail` is designed to convey an error message if something goes wrong during the operation of the IPXE. It first calls the `ipxe_header` function, logs the error message, and then displays an error in the IPXE shell. This includes the error message passed to the function, a sleep command to pause execution for 10 seconds, and then a reboot. The function then ends using the `exit` command.

2.0.682 Technical Description

- **Name:** `ipxe_cgi_fail`
- **Description:** Generates and displays an error message when IPXE encounters an issue.
- **Globals:** None.
- **Arguments:** [$1: Error message to be displayed in IPXE and logged]
- **Outputs:** An error message in IPXE shell, and a log entry.
- **Returns:** None. The function ends with an `exit` command after displaying the error message, pausing for 10 seconds, and rebooting.
- **Example usage:**

```
ipxe_cgi_fail "Unable to connect to the server"
```

2.0.683 Quality and Security Recommendations

1. Always ensure that the function is supplied with a meaningful and useful error message to help with debugging.
2. It might be beneficial to include some environment information in the log message or the error output to provide additional debugging context.
3. Exercise caution when executing from an untrusted context or with unsanitized inputs, as the log message could potentially expose sensitive data.
4. Any usage of this function results in a reboot. Ensure that this is an acceptable behavior and that it does not unexpectedly disrupt other processes or tasks. If not, consider modifying the function to provide alternative execution paths.
5. Regularly review the logs for errors to inform improvements to the system as a whole and increase overall robustness.

2.0.684 `ipxe_cgi_fail`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 98eb961ae3ed7dfffc7f4ae68cd1c88aa5f47672ee53b71b29f0428922e8efaa

2.0.685 Function overview

The function `ipxe_cgi_fail()` is utilized to handle failure during the Implicit PXE (iPXE) process. This function generates an iPXE header, sends an error message to the

hp's log, reports the error in the iPXE code to be interpreted by iPXE supporting software, waits for 10 seconds, then reboots the system.

2.0.686 Technical description

- **Name**: ipxe_cgi_fail
- **Description**: This function is intended to handle failures during the iPXE process. Upon invocation, it creates an iPXE header, logs an error message, alerts the user about the failure, waits a bit, then reboots the system.
- **Globals**: None.
- **Arguments**:
  - `$1: cfmsg`: This is the error message to be logged and displayed.
- **Outputs**: An iPXE formatted error message, including the input error message.
- **Returns**: Nothing. The function does not have a return statement, but it does call exit, terminating the script it's within.
- **Example usage**: `ipxe_cgi_fail "Network boot failed"`

2.0.687 Quality and security recommendations

1. **Input Validation**: Implement input validation on `$1` to check if it is set and isn't an empty string before proceeding with the rest of the function. This would make the function more robust against erroneous invocations.
2. **Error handling on `exit`**: The script terminates abruptly with `exit`. It's recommended instead to return an error code, and let the main section of your script decide how to handle the error.
3. **Message Standardization**: It's suggested to use standardized error codes and messages to make troubleshooting more efficient.
4. **User Instructions**: Since the error causes the system to reboot, provide more information to the user regarding what they should do after the reboot.
5. **Securing Logging**: Ensure only authorized and authenticated applications or services can write to the log file. This prevents unauthorized modifications which could lead to misinterpretation of system states.

2.0.688 `ipxe_configure_main_menu`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 7e0f06b766ddbf72d17a72c59b077064b8058b4fa4dd5d3f6e83da96e88a361a

2.0.689 Function overview

The `ipxe_configure_main_menu` function is part of a shell script that is used to set a configuration menu on iPXE for a server or host. When called, it creates an interface with multiple menu options, such as configuration options or choices to enable or disable

forced installations. The function then fetches logs, processes selected menu items, and chains them with a cgi url.

2.0.690 Technical description

- Function name: `ipxe_configure_main_menu`
- Description: The function delivers a configuration menu on iPXE for a server or host. It generates a user interface with multiple options, processes chosen menu items and delivers them via a CGI URL.
- Globals:
  - TITLE_PREFIX
  - CGI_URL
  - mac
- Arguments: Not applicable.
- Outputs: The function prints a configuration menu interface to stdout.
- Returns: No explicit return value; the function's result is based on its side effect of displaying a menu and possibly logging messages.
- Example usage:

```
ipxe_configure_main_menu
```

2.0.691 Quality and security recommendations

1. Validate Input: Always validate input, especially if getting input from users, to avoid potential command injections that could lead to security vulnerabilities.
2. Use ShellCheck: Utilize ShellCheck, a linting tool for shell scripts, to detect any syntax errors, wrong command or argument usage etc.
3. Documentation: Comment your code more thoroughly for maintainability.
4. Error Handling: Implement proper error or exception handling to prevent your script from abruptly stopping in case of unusual or unexpected inputs or situations.
5. Security: Use SSL or TLS and if possible, avoid using `CGI_URL` in plain text. Instead, consider applying security measures such as encryption to protect sensitive data.

2.0.692 `ipxe_configure_main_menu`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 7e0f06b766ddbf72d17a72c59b077064b8058b4fa4dd5d3f6e83da96e88a361a

2.0.693 Function Overview

The function `ipxe_configure_main_menu()` plays a vital role in system configuration, specifically in setting up the main menu for the host machine. The main menu is

delivered in circumstances where the overall system cluster is configured, yet the host itself is not. It includes options for changing host configuration, viewing the host and cluster configuration, network recovery boot, entering a rescue shell, booting from local disk, rebooting the host, and options to allow and disable management by HPS, amongst other functionalities.

2.0.694 Technical Description

- **Name:** `ipxe_configure_main_menu()`
- **Description:** This function is utilized to configure the main menu for a host machine when the system cluster is configured but the host is not. It includes numerous functionalities that aid in system setup, administration, and troubleshooting.
- **Globals:** None.
- **Arguments:** None.
- **Outputs:** It outputs a menu consisting of multiple host options that can be selected by users.
- **Returns:** It does not return any specific value.
- **Example Usage:** `bash ipxe_configure_main_menu`

2.0.695 Quality and Security Recommendations

While the function is robust, there are potential areas of enhancement and precaution:

1. The function has hard-coded values which can be extracted as variables or constants at the start of the function. Hard-coded values are generally considered a bad programming practice as they can lead to difficulties during system maintenance and scalability issues.
2. It currently does not handle error cases. For instance, if the `host_config` or `imgfetch` commands fail, there are no error handling mechanisms in place. In future implementations, consider implementing mechanisms to handle such failures gracefully.
3. Log messages are being sent over HTTP by the `imgfetch` command which could potentially expose sensitive information. Consider encrypting important information or using HTTPS for secure communication.
4. The function does not have any input validation. To reduce the occurrence of bugs or unexpected behaviour, validate any inputs that come into the program.
5. A 'test mode' could be beneficial. This would be a way to run the function without it affecting the production environment, allowing you to confirm that it behaves as expected under various conditions.
6. The script could be refactored to be more modular, enhancing readability, maintenance, and potentially identifying areas for optimization.

2.0.696 ipxe_goto_menu

Contained in lib/functions.d/ipxe-menu-functions.sh

Function signature: 49847df81dbbc4d2221bff922ec99935856f612fa6c68856d80a8ee4f35615de

2.0.697 Function overview

The ipxe_goto_menu is a Bash function that refreshes the ipxe environment and directs the user towards a chosen menu or to the main menu by default. This is achieved by building a chain request to the ipxe server with the desired menu item as a parameter. As an element of a network booting solution, the scope of the function is to allow adjustments in the booting process through menu interaction.

2.0.698 Technical description

- Name: ipxe_goto_menu
- Description: The function refreshes the ipxe environment and navigates to a chosen menu (defaulting to the main menu if not specified).
- Globals: This function does not use/globalize any variables.
- Arguments:
  - $1: This argument represents the menu that the function navigates towards. Defaults to init_menu if not provided.
- Outputs: The function initializes an ipxe header and outputs ipxe commands to free loaded images (imgfree) and create a chain loading configuration (chain).
- Returns: Does not return any particular value, simply executes commands.
- Example usage:

```
ipxe_goto_menu          # Will navigate to "init_menu" by default
ipxe_goto_menu "custom_menu"  # Will navigate to "custom_menu"
```

2.0.699 Quality and security recommendations

1. Introduce error handling for the ipxe_header function call inside - it's potentially susceptible to failures which are not currently captured.
2. Check the validity of CGI_URL before using it. Currently, if CGI_URL is incorrect, the function will fail without any error messages. Customizable error messages will greatly improve troubleshooting.
3. Validate the MENU_CHOICE argument to ensure it corresponds to an existing menu. This can prevent erroneous chains.
4. For added security, sanitize the MENU_CHOICE variable to prevent potential command injection attacks.

2.0.700 ipxe_goto_menu

Contained in lib/functions.d/ipxe-menu-functions.sh

Function signature: 49847df81dbbc4d2221bff922ec99935856f612fa6c68856d80a8ee4f35615de

2.0.701 Function overview

The function `ipxe_goto_menu` is used to refresh iPXE and navigate to the main menu or any specified menu. The name of the desired menu is passed as an argument to the function. If no name is provided, it defaults to `init_menu`. The function then calls upon `ipxe_header` and uses ipxe commands to free any previously allocated images and to initiate a chain loading process from a specific URL containing the defined menu name.

2.0.702 Technical description

- **name**: `ipxe_goto_menu()`
- **description**: The function refreshes iPXE and navigates to either the main menu or to a specified menu.
- **globals**: None
- **arguments**: `$1:MENU_CHOICE` - The name of the menu to navigate to. If none is specified, it defaults to `init_menu`.
- **outputs**: Displays whatever is output by the `ipxe_header` function, an `imgfree` line, and the chain loaded URL.
- **returns**: No explicit return value. It changes the state of the running iPXE.
- **example usage**: `ipxe_goto_menu init_menu`

2.0.703 Quality and security recommendations

1. In the function, consider checking the validity of the `MENU_CHOICE` argument to make sure it corresponds to a valid menu name.
2. Escape special characters in `MENU_CHOICE` to prevent potential command injection vulnerabilities.
3. Always double-check the CGI_URL value - it should only contain trusted URLs.
4. Ideally, the function should provide feedback when the chaining process fails.
5. For robust error handling, the function could benefit from a separate error handler providing meaningful error messages.

2.0.704 `ipxe_header`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 173b539694912c98423ee15feb900097775a6a7016f10506826ba220e09c060d

2.0.705 Function overview

The `ipxe_header()` function is a Bash function designed to prevent boot failures by sending a pxe header. It first calls the `cgi_header_plain` function, and then

sets up several variables to be used in IPXE scripts. These variables include STATE, CLUSTER_NAME, and TITLE_PREFIX. The function then prints out a block of text which is used to update the iPXE log message and to display system information.

2.0.706 Technical description

- **Name**: ipxe_header
- **Description**: This Bash function prevents boot failures by transmitting a PXE (Preboot Execution Environment) header. It sets up several environment variables used in iPXE scripts, and prints out a block of text output which includes the log message and system information.
- **Globals**:
  - VAR: Various environment variables such as `state`, `cluster_name` and `title_prefix`.
- **Arguments**: N/A
- **Outputs**:
  - Prints an iPXE script which primarily serves to log messages and display system details.
- **Returns**: Null
- **Example usage**:

```
source ipxe_header.sh
ipxe_header
```

2.0.707 Quality and security recommendations

1. Ensure the function is used in a secure environment where potential abusers cannot manipulate the value of the `mac` variable.
2. The function is dependent on `cgi_header_plain` function, always make sure that this function is present and working correctly before using `ipxe_header`.
3. The function does not seem to have any error checking. Always ensure that the functions `host_config` and `cluster_config` are not failing and not returning any unexpected results.
4. The global variable `$CGI_URL` on `imgfetch` line should be carefully managed to avoid potential security vulnerabilities like command injection. Make sure to use proper escaping or better yet, refactor and avoid using direct variable substitution.
5. In general, it is recommended to always initialize local variables and check their values before using them to avoid unexpected behavior or potential errors.

2.0.708 `ipxe_header`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 173b539694912c98423ee15feb900097775a6a7016f10506826ba220e09c060d

2.0.709 Function Overview

The `ipxe_header` function provides a mechanism to send a network boot payload or Preboot eXecution Environment (PXE) header. This ensures that the boot process does not fail due to missing network information. The function sets up variables to be used in IPXE scripts, retrieves the system and cluster configuration, sets a title prefix for the payload, and includes log information that conveys the details about the client and the function call used.

2.0.710 Technical Description

Name: `ipxe_header`

Description: This function is used to send a PXE header, to prevent boot failure. It also sets up several variables for use in IPXE scripts, and sends useful logging information.

Globals:

- `VAR: TITLE_PREFIX` - Prefix for the payload title, which includes cluster name, client IP, MAC address, and the function name.

Arguments:

- `mac`: The MAC address of the host machine.
- `CLUSTER_NAME`: The name of the current cluster.

Outputs:

- Standard output of a iPXE network boot payload

Returns:

- The function does not return a specific value

Example Usage:

```
ipxe_header
```

2.0.711 Quality and Security Recommendations

1. Use shellcheck or another linting tool to ensure that the code adheres to best practices about variable usage, command substitution and so on.
2. Be careful when dealing with unknown values as part of the function, to ensure no unanticipated behavior or erroneous output occurs.
3. Handle potential exceptions that may occur while fetching the host configuration.
4. Make sure to sanitize all inputs to prevent command injection attacks.
5. Use secure communication to fetch the host and cluster configuration.
6. Ensure logging information does not contain any sensitive data that might expose the system to security risks.

2.0.712 ipxe_host_audit_include

Contained in lib/functions.d/ipxe-menu-functions.sh

Function signature: e8e0c76631f146b481fc585b6e35f98c7c60fd0fdefbc65eeb7573a8b6bd544e

2.0.713 Function Overview

The ipxe_host_audit_include function is designed for auditing a host's details in an iPXE environment. This function gathers system information including manufacturer, product, serial number, memory size, build architecture, and platform. It also collects network information – IP address, gateway, DNS, and DHCP server. The function fetches this data using the imgfetch command, sending the information to a specific URL.

2.0.714 Technical Description

- **Name:** ipxe_host_audit_include
- **Description:** The function collects host details such as manufacturer, product, serial number, memory size, build architecture, and platform and sends them to a remote URL. It also gathers network and SMBIOS data.
- **Globals:**
  - manufacturer: System manufacturer
  - product: Product name
  - serial: Serial number
  - memsize: Memory size
  - buildarch: Build architecture
  - platform: System platform
  - CGI_URL: The URL to send the audit data to
- **Arguments:** N/A
- **Outputs:** Sends system, network, and SMBIOS data to a remote URL. If the imgfetch command fails, it outputs a failure message.
- **Returns:** N/A
- **Example Usage:**

```
source ipxe_host_audit_include.sh
ipxe_host_audit_include
```

2.0.715 Quality and Security Recommendations

1. Validate user input: Ensure that the CGI_URL is a valid and trusted source before sending any data.
2. Error handling: Determine how the function should proceed if the imgfetch command fails or if certain data cannot be obtained.
3. Commenting: Use descriptive comments to further explain what each section of the function does.

4. Global variables: Avoid using global variables where possible as they may be overwritten by other parts of the script.
5. Data security: Ensure that sensitive information such as the system's serial number is protected and only shared with trusted sources.

2.0.716 `ipxe_host_audit_include`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: e8e0c76631f146b481fc585b6e35f98c7c60fd0fdefbc65eeb7573a8b6bd544e

2.0.717 Function overview

This function, `ipxe_host_audit_include()`, is created for use within other functions and hence does not send a header. Its purpose is to facilitate a system audit with validation. It checks whether various system variables (i.e., manufacturer, product, serial, memsize, buildarch, platform) are set, and if they are not, it sets them to default values. Following this, it fetches and stores the 'audit_data', 'net_data', and 'smbios_data' by making HTTP GET requests. If these fetch requests fail, corresponding error messages are printed.

2.0.718 Technical description

- **Name:** ipxe_host_audit_include
- **Description:** This function is used for system auditing, setting default values for system variables if they are not set. It then fetches data about the system (audit_data), network (net_data) and SMBIOS (smbios_data) and stores it.
- **Globals**: [CGI_URL: The URL the data is fetched from]
- **Arguments:** None
- **Outputs:** Audit data about the system, network and SMBIOS. If fetch requests fail, it outputs "Audit failed", "Network failed", "SMBIOS failed", respectively.
- **Returns:** None, this function does not have a return statement.
- **Example usage:** This function is an include, i.e., it is to be used within other functions. Hence, it will not be called directly as a standalone function.

2.0.719 Quality and security recommendations

1. Protect against HTTP errors - Currently, if any HTTP request fails, the function simply outputs an error message and execution continues. Instead, the function could halt execution or attempt recovery upon encountering an HTTP error.
2. Verify URIs - No verification is currently done on the URIs before they are passed to `imgfetch`. Implementing URI verification could help avoid potential issues.

3. Protect against variable manipulation - It would be advisable to implement checks to ensure that the globals used in this script (like ${CGI_URL}) are set and have not been manipulated.
4. Log fetch failures - Instead of just printing the error messages when fetch requests fail, these could be logged to a system log file for future review.

2.0.720 `ipxe_host_configure_menu`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: ef7be378c4f9da45c3b5dd7aa5b9049fb864669e400c2fbb36e8dc3c81609b0a

2.0.721 Function overview

The function `ipxe_host_configure_menu` is part of the `ipxe` program process. This is used for executing network booting and template scripts to set up a more expanded and flexible range of installation options during the booting process.

2.0.722 Technical description

Name: `ipxe_host_configure_menu`

Description: This is a bash shell function aimed at setting up a configuration menu when booting up the system on IPXE. It provides a set of different profiles which users can choose from depending on what they wish to implement on the booting system - options range from default profiles, thin compute host to storage cluster host.

Globals: N/A

Arguments: The function does not require any arguments.

Outputs: It generates a booting menu interface where users can select a variety of Unix profiles.

Returns: The function does not have explicit return outputs.

Example usage: `ipxe_host_configure_menu`

2.0.723 Quality and security recommendations

1. Include a more detailed error messages for debugging purposes. This is significant for developers to easily locate quick fixes for any function malfunctions.
2. Incorporate validation checks within the function. This is crucial for securing any passed arguments, ensuring no unexpected or harmful data is processed by the function.
3. Implement a fallback default selection, in case the chosen configuration fails for any reason.

4. Make sure the function works well with other functions in the same script and does not inadvertently modify any global variables.
5. Lastly, always ensure to test the function exhaustively to affirm its correctness in various scenarios. This is critical to ensure its flexibility, reliability and efficiency, particularly in unexpected conditions and edge cases.

2.0.724 `ipxe_host_configure_menu`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: ef7be378c4f9da45c3b5dd7aa5b9049fb864669e400c2fbb36e8dc3c81609b0a

2.0.725 Function overview

The function `ipxe_host_configure_menu` presents the user with a customizable installation menu. This menu allows the user to select an installation option based on the machine's MAC address. The menu also displays relevant installation options based on whether the cluster has an Installed Storage Cluster Host (SCH). The selected option then gets processed and logged for future reference.

2.0.726 Technical description

- **name:** `ipxe_host_configure_menu`
- **description:** This function generates and manages an interactive installation menu that is responsive to the cluster's current configuration. It leads to the installation of either Thin Compute Host (TCH) or Disaster Recovery Host (DRH) or Storage Cluster Host (SCH) based on the user's selection.
- **globals:** [`TITLE_PREFIX`: Prefix of the title for the generated menu, `CGI_URL`: URL to the CGI script that processes the selected menu item]
- **arguments:** None
- **outputs:** An interactive menu printed to stdout. Logs the selected menu item to an external log file.
- **returns:** Nothing. The function's primary operation is side-effected.
- **example usage:** `ipxe_host_configure_menu`

2.0.727 Quality and security recommendations

1. Add input validation: Currently, this function does not validate menu selection input. Implementing validation could enhance the function's reliability and security.
2. Handle errors explicitly: The function attempts to fetch an image, but if it fails, it simply prints "Log failed". It could enhance error tracking by throwing an exception or outputting more detailed error information.
3. Implement logging: Incorporate a more production-level logging system, instead of only using the `imgfetch --name log` command.

4. Use secure command options: When executing `chain --replace`, ensure that the URL in `CGI_URL` is safely encoded to prevent command injection attacks.
5. Code comments: Add comments in the code to improve readability and maintainability of the code.

2.0.728 `ipxe_init_handler`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 0c18738a58080d4974ef58caa0d3be0a23ce55dcc5fec19ba15765643b82e245

2.0.729 Function overview

The `ipxe_init_handler` function is a Bash shell function. It performs initial setup for the Intelligent Platform Management Interface (IPXE) based on a host node's MAC address. The function first logs that initialization is requested, detects the architecture, and verifies if the host configuration exists. If the configuration is absent, it initializes a new one. The function then gets the machine's current state. If that state includes RESCUE mode active, the function calls `ipxe_network_boot` and logs an informational message before returning zero. The control flow switches upon the `state`. Each state logs a message and performs specific actions, but if the state isn't known, the function logs an error and returns one.

2.0.730 Technical description

- Name: `ipxe_init_handler`
- Description: This function initializes an iPXE setup based on a host node's MAC address.
- Globals: None.
- Arguments:
  - `$1`: This is the MAC address of the machine to initialize.
- Outputs: There are several log information entries during the function's execution, depending on the input variables.
- Returns:
  - `0` if the function executes successfully or RESCUE mode is active.
  - `1` if the `state` is unknown or not set.
- Example usage:

```
ipxe_init_handler "08:00:27:53:8b:38"
```

2.0.731 Quality and security recommendations

1. Make sure MAC addresses are validated before they are passed in to prevent possible Injection attacks.

2. In the comment at the start of the script, it is noted architecture detection is not yet designed to recognize the running architecture - it is hardcoded to “x86_64”. Implement a mechanism to properly detect the architecture.
3. The function relies on environment variables, which could be a vulnerability if they’re not handled securely. Validate and sanitize environment variables before use.
4. Using `$state` in error logging can expose internal data, replace or sanitize `$state` output in the error log.
5. Provide a proper cleanup and error handling for situations where the function does not meet expectations.

2.0.732 `ipxe_network_boot`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: e2e2b58a92e9af696185f032c9e7372a47cee6530ad51ea9189fc8f257805669

2.0.733 Function overview

The function `ipxe_network_boot` is a Bash function which is used to boot a Network Interface Controller (NIC) by using the Internet Protocol network booting method (iPXE). It accepts the MAC address of the NIC, obtains the operating system (OS) name using the MAC address, logs the booting process, and calls the function corresponding to the OS name, provided that function is declared. If no such function is declared, an error message is returned.

2.0.734 Technical description

Function Name: ipxe_network_boot

Description: This function bootstraps a network boot using iPXE for the provided MAC address.

Globals: None

Arguments: - `$1`: The MAC address of the NIC to be booted.

Outputs: Debug logs on standard output showing the OS being net booted.

Returns: Calls the respective iPXE boot function based on the OS name. If the function does not exist, it terminates with an error message.

Example Usage: `ipxe_network_boot "00:11:22:AA:BB:CC"`

2.0.735 Quality and security recommendations

1. Always use quotes around variable names in bash to avoid errors when they contain spaces.

2. Implement additional error checking to confirm if a valid MAC address is passed as an argument.
3. Ensure that logging does not expose sensitive data, which may potentially be exploited.
4. Make sure that the function only allows booting the desired OSs and deny others for security reasons.
5. It is recommended to implement checks to make sure that only the authorised users can use this function.
6. Always check return values and error messages for all operations.

2.0.736 `ipxe_reboot`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: c2e2fb28eaa8f9898d8b5cb181b2b278c884005d1a98813c38797c3225f12b52

2.0.737 Function overview

The function `ipxe_reboot` is designed to print a log message indicating a reboot request, followed by the actual reboot request. Initially, it defines a local variable `MSG` and assigns it the value of `$1`, the first argument passed to the function. If the `MSG` is not null, it is printed, and regardless, a standard rebooting message is printed. Finally, it waits for 5 seconds and then triggers the reboot.

2.0.738 Technical description

- **Name:** `ipxe_reboot`
- **Description:** This function is used to log a reboot request and then reboot the system. This includes constructing a header using `ipxe_header`, printing a custom reboot message if provided, and echoing a standard “Rebooting…” message. It then instructs the system to sleep for 5 seconds before rebooting.
- **Globals:** No globals are used explicitly in this function.
- **Arguments:**
  - `$1:` `MSG`: An optional message to be displayed during the reboot process.
- **Outputs:**
  - If a message is provided as an argument (`MSG` is not null), it is printed.
  - A standard “Rebooting” message is always printed.
- **Returns:** The function doesn’t return a value.
- **Example usage:**

```
ipxe_reboot "Scheduled system reboot"
```

This will log a message “Reboot requested Scheduled system reboot”, output the provided message, and then the system will reboot after a pause of 5 seconds.

2.0.739 Quality and security recommendations

1. Buffer Overflow Protection: To avoid any risk of buffer overflow, the function should previously validate the length of the `$1` input argument before assigning it to the MSG variable.
2. Error Handling: If the reboot command fails to execute, this function doesn't handle it. This can be improved by adding error handling routines or exit codes checking for each command in the function.
3. Secure Logging: The function simply logs the message, "Reboot requested $MSG". To ensure secure logging, it should also log the identity of the user or process that initiated the reboot, and timestamp each log entry.
4. Documentation: Each step and the overall purpose of the function should be properly documented in the code.

2.0.740 `ipxe_reboot`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: c2e2fb28eaa8f9898d8b5cb181b2b278c884005d1a98813c38797c3225f12b52

2.0.741 Function overview

The function, `ipxe_reboot`, accepts a single argument, MSG, logging that a reboot was requested with MSG as a parameter, outputs specific headers using the `ipxe_header` function, and then echoes MSG. If MSG is not an empty string, it echoes that the system is rebooting, puts the system to sleep for 5 seconds, and then reboots the system.

2.0.742 Technical description

- **Name**: `ipxe_reboot`
- **Description**: This function logs an info message showing that a reboot is requested, outputs headers using another function `ipxe_header`, checks if MSG (first input argument) is not a null string and echoes it if true. It then echoes "Rebooting…", puts the system to sleep for 5 seconds and then reboots the system.
- **Globals**: None
- **Arguments**:
  - `$1 (MSG)`: The message to be logged and echoed just before the reboot command. It's optional, and if it's null or not defined, it won't be used.
- **Outputs**: Echoed statements to the standard output for logging and status updates.
- **Returns**: No value is returned as the terminal will be closed upon successful function execution because of the `reboot` command.
- **Example usage**: `ipxe_reboot "System updates are completed"`

2.0.743 Quality and security recommendations

1. Always make sure to validate the input parameters. Even though this function does not explicitly use user-provided inputs, it's still a good practice.
2. The `reboot` command is a powerful system command. Ensure this function is only accessible to and executable by authorized users and applications to prevent misuse.
3. There are no command success/failure checks in the function. Consider adding error handling or command result checks to make the function more robust.
4. In an environment where the system log is reviewed or parsed, consider standardizing the log format to make the logs more readable.
5. The function depends on another function, `ipxe_header`. Ensure this dependant function is robust, secure, and available where `ipxe_reboot` is used to avoid runtime errors.
6. Avoid hard-coding values such as the sleep duration. It would be a better practice to make such values configurable.

2.0.744 `ipxe_show_info`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 6a7c0b8558a30a72d8737d1e4f7ee6666cab142570d39fe937cbbc8ab4e87ee2

2.0.745 Function overview

The `ipxe_show_info()` function in Bash is responsible for displaying information related to iPXE (open source boot firmware), over different categories, that the user can choose. The categories include system-specific data, such as MAC, IP, Hostname, and other variables, as well as cluster configuration, host configuration, and HPS paths.

2.0.746 Technical description

- **name**: `ipxe_show_info`
- **description**: This function displays a menu to the user from which they can choose to see iPXE host data, cluster configuration, host configuration, or HPS paths. The function makes use of the HERE document feature in bash to create on-screen menus.
- **globals**: `HPS_CLUSTER_CONFIG_DIR`, `CGI_URL`, `HPS_CONFIG`
- **arguments**: `$1: category (which kind of information to show)`
- **outputs**: Depending on the chosen category, this function outputs various pieces of data to the user. This could be system information like the MAC address, IP and hostname, or more specific configuration details for a cluster or host.
- **returns**: None, it only prints to stdout.

- **example usage**: `ipxe_show_info show_cluster`

2.0.747 Quality and security recommendations

1. Make sure variable names are self-explanatory and properly defined.
2. Always ensure that the function doesn't use or print any sensitive data.
3. Check for the existence of files before attempting to read from them.
4. Try to avoid using global variables within a function, to maintain code flexibility and avoid possible side-effects.
5. Handle potential errors gracefully, by using some type of error catching mechanism. For example, there could be an error clause where it handles situations where the passed category doesn't exist.
6. Make sure that any potential threat regarding command injections is mitigated. For instance, escape any special characters in user inputs if such inputs are used to form commands.
7. Ensure that the function is properly commented, not only for others to understand but also for the ease of mitigating new bugs and issues.
8. Validate and sanitize all user inputs. It's an important principle of security to never trust user input.
9. Regularly update and patch the software when new versions are available.

2.0.748 `_is_tty`

Contained in `lib/functions.d/system-functions.sh`

Function signature: 53fbf6cc003bc7d9b4614fc9d372f3471ed01447874f4db99da2816eb3cb7e69

2.0.749 1. Function Overview

The Bash function `_is_tty` is designed to check whether the standard input (stdin), standard output (stdout) or the standard error (stderr) of the current terminal is attached to a tty (terminal).

2.0.750 2. Technical Description

- **Name:** `_is_tty`
- **Description:** The function checks if standard input (stdin), standard output (stdout), and standard error (stderr) are currently attached to a tty (terminal). This is achieved by using the `-t` test which returns true if the file descriptor is open and associated with a terminal.

- **Globals:** None
- **Arguments:** None
- **Outputs:** No explicit output. Internally the function returns with a status of 0 if any of the standard I/O streams are attached to a tty, or with a non-zero status otherwise.
- **Returns:** It returns true if at least one of stdin, stdout, or stderr is attached to a terminal. Otherwise, it returns false.
- **Example Usage:**

```
if _is_tty; then
    echo "We're in a tty terminal."
else
    echo "We're not in a tty terminal."
fi
```

2.0.751 3. Quality and Security Recommendations

1. Do not use this function if data privacy is your concern. In shared systems, other users can read or write to this terminal if they know its tty device file.
2. Always check the result of the function to handle the non-interactive environments appropriately.
3. It's highly recommended to handle the return status of this function properly to prevent faults in scripts that depend on a tty terminal.
4. Since the function has no arguments or global side-effects, it's safe to use this in any part of your scripts. But be aware that it only checks the state of the terminal at the time the function is called, not continuously.

2.0.752 `keysafe_cleanup_expired`

Contained in `lib/functions.d/keysafe_functions.sh`

Function signature: 3206e50a2d55d74e70a68067e5cb3e041c93edf9f4d93411da00ecaa4c3535d8

2.0.753 Function overview

The `keysafe_cleanup_expired` function iterates through the files (referred to as tokens) in a specific directory, detects the expired ones, removes them, and logs the number of removed tokens. This function is part of a larger system (possibly a key or token management system) where tokens have finite lifetimes and need to be cleaned up once they expire to prevent stale data accumulation.

2.0.754 Technical description

- **Name**: `keysafe_cleanup_expired`

- **Description**: This function cleans up expired tokens. It gets the directory where the tokens are stored, iterates through all tokens, and removes the ones that have expired. Upon completion, it logs the count of tokens that were removed.
- **Globals**: None.
- **Arguments**: None.
- **Outputs**: If any tokens were removed, it logs to STDERR in the following format: "Cleaned up [count] expired token(s)". Here [count] is the number of removed tokens.
- **Returns**: 0 (upon successful completion), 1 (in case of failure to obtain the token directory)
- **Example usage**:

  `keysafe_cleanup_expired`

2.0.755 Quality and security recommendations

1. The function should handle errors in a more robust way. Currently, if it fails to get the keysafe directory, it merely returns 1. However, there aren't any checks or exception handling afterward.
2. Security-wise, it's crucial to validate whether the function has the necessary permissions to manipulate the token files (read, delete).
3. The function leverages arbitrary file inclusion with the 'source' command, which can lead to security issues if arbitrary user input can influence the files that are included.
4. It's recommended to implement some form of logging that indicates the exact tokens (files) that were cleaned up.
5. It's useful to sanitize any inputs and consider potential race conditions in circumstances where tokens might be added or removed by other processes while this function executes.

2.0.756 `keysafe_handle_token_request`

Contained in `lib/functions.d/keysafe_functions.sh`

Function signature: c84ad9d04dca1d7aa2bb79eee2302c774f89404291568bc324a376f3d3c8f37d

2.0.757 Function Overview

The primary purpose of the `keysafe_handle_token_request` function is to manage and handle token requests. This function accepts a MAC address and a purpose as parameters. It validates these parameters and in case one of them is not provided, a warning message is logged and the function returns an error code. Then, the function fetches the node id by using the provided MAC address. If the node id couldn't be

determined, the function sets it to “unknown” and continues its operation. The next step is issuing a token. If the token is issued successfully, it is returned by the function. Otherwise, an error message is logged and shown.

2.0.758 Technical Description

- **name:** keysafe_handle_token_request
- **description:** This function manages token requests by validating inputs, handling exceptions and finally issuing a token.
- **globals:** [mac: MAC address, purpose: The purpose for which the token is being requested]
- **arguments:** [$1: MAC address, $2: Purpose]
- **outputs**: On successful completion, the function prints the issued token. On failure, it logs an error message and returns an error code.
- **returns**: If function is successful, it returns 0. If MAC address or purpose is missing, it returns 1 or 2 respectively. If unable to issue the token, it returns 3.
- **example usage:** keysafe_handle_token_request “aabbccddeeff” “token_request”

2.0.759 Quality and Security Recommendations

1. Implement input sanitation for the ‘mac’ and ‘purpose’ parameters to avoid potential command injection vulnerabilities.
2. Improve error handling by providing meaningful error messages for every return code.
3. Implement further checks to avoid issuing tokens to invalid MAC addresses or nodes.
4. The function might benefit from a restructure to reduce its complexity and improve the readability of the code.
5. It’s recommended to employ debug logging at the start of the function execution to aid in troubleshooting potential issues.

2.0.760 `keysafe_issue_token`

Contained in `lib/functions.d/keysafe_functions.sh`

Function signature: a535c705c6f66c5f6bb4cb03f77f0a1320e62fa27e687ab3cf0e601508b20886

2.0.761 Function overview

The Bash function `keysafe_issue_token()` generates a unique token in two modes, open and secure, based on the provided client MAC address and the intended purpose. It saves the generated token along with some metadata in a file on the keysafe directory and then returns the token back to the caller. If no node id is provided as the third parameter when the function is called, it defaults to “unknown”.

2.0.762 Technical description

- **Name:** `keysafe_issue_token`
- **Description:** This function generates and returns a unique token after validating the arguments (client_mac and purpose). The function either uses a UUID for open mode or Biscuit token generation for secure mode (not yet implemented). It then records the information about the issued token.
- **Globals:** [`HPS_CLUSTER_CONFIG_BASE_DIR: This holds the base directory for cluster configuration`, `HPS_KEYSAFE_MODE: Holds the keysafe mode which can be open or secure`]
- **Arguments:** [`$1: client_mac`, `$2: purpose`, `$3: node_id (optional, defaults to "unknown")`]
- **Outputs:** Prints the generated token to stdout, or logs error messages to stderr when problems occur.
- **Returns:** `0 if successful`, `1 if keysafe directory not found`, `2 if unable to create token file`, `3 if required arguments are missing`, `4 if secure mode is not yet implemented`, `5 if invalid keysafe mode`.
- **Example usage:** `keysafe_issue_token "MAC_ADDRESS" "PURPOSE" "NODE_ID"`

2.0.763 Quality and security recommendations

1. Improve error handling: Add more detail to error outputs, so that the specific error condition can be readily identified.
2. Implement secure mode: The secure mode for token generation has not been implemented and should be as soon as feasible.
3. Add validation for modes: Validate that only known modes (“open”, “secure”) can be assigned to `HPS_KEYSAFE_MODE`.
4. Exclude open mode in production: Do not use the open mode in a production environment as it is insecure and meant only for prototyping.
5. Store tokens securely: Depending on the function’s usage, consider storing the generated tokens in a more secure or encrypted manner.

2.0.764 `keysafe_validate_token`

Contained in `lib/functions.d/keysafe_functions.sh`

Function signature: cabc1a12f681dbd43df7a58a8cdb46b267a8bb8a11fcda90cb24e16e5ad3a669

2.0.765 Function Overview

`keysafe_validate_token` is a function in Bash, designed to validate the token passed from keysafe. It validates the token against the expected purpose and also checks

the token expiration. If the token is valid, it gets consumed and removed immediately. In case the provided token is invalid or expired or doesn't match the expected purpose, the function provides relevant error messages.

2.0.766 Technical Description

- **Name:** `keysafe_validate_token`
- **Description:** This function validates the keysafe token for its authenticity, expiration, and the purpose for which it was generated. It reads the token data from a file and deletes the file if the token is valid.
- **Globals:** None
- **Arguments:**
  - `$1` (token): The token passed to the function to be authenticated.
  - `$2` (expected_purpose): The expected value of the token purpose.
- **Outputs:** Error or warning messages.
- **Returns:** Returns error codes. (1: Keysafe directory failure, 2: Invalid or already consumed token, 3: Token expired, 4: Token purpose mismatch, 5: Token argument missing)
- **Example Usage:**

```
keysafe_validate_token "$your_token" "your_expected_purpose"
```

2.0.767 Quality and Security Recommendations

1. Add error handling if the `source` command fails to read the token metadata.
2. Always validate user input. Ensure that both the token and purpose values are sanitized before they are processed in the function.
3. It's good practice to delete the used token immediately, as shown in this function. However, a failsafe mechanism should be in place if deleting the file fails.
4. In the interest of robustness, consider implementing logic to handle when `get_keysafe_dir` function fails to retrieve keysafe directory.
5. Always log error messages to help with troubleshooting. However, avoid logging sensitive data such as the actual token value.

2.0.768 `list_cluster_hosts`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: be089e20c3a7eb6f1c96e5243449b6bffb67b4c25d9062593c07c1c4f6b55ffb

2.0.769 Function overview

The function `list_cluster_hosts()` is used to list the MAC addresses of the cluster hosts. It accepts a cluster name as an argument, gets the cluster directory, determines the hosts directory depending on the cluster name, and verifies the existence of this directory. Then it lists all .conf files in the directory, extracts the MAC addresses from them, and prints them out. If no argument is passed, it assumes the active cluster directory.

2.0.770 Technical description

- **Name**: `list_cluster_hosts()`
- **Description**: Lists the MAC addresses of the hosts in a given cluster.
- **Globals**: None.
- **Arguments**:
  - `$1: cluster_name` - The name of the cluster. If omitted, uses the active cluster.
- **Outputs**:
  - Lists the MAC addresses of the hosts in the identified cluster.
  - Logs errors and debug information to the console.
- **Returns**:
  - `0` if everything is fine,
  - `1` if it cannot determine the active cluster hosts directory or cannot get the directory for the specified cluster.
- **Example Usage**:
  - `list_cluster_hosts myCluster`
  - `list_cluster_hosts`

2.0.771 Quality and security recommendations

1. Validate the provided cluster name to ensure it matches expected formatting and values.
2. Consider implementing error handling for the cases where the .conf files cannot be read or are improperly formatted.
3. Introduce strict mode (`set -euo pipefail`) at the top of your script. This forces you to handle unintended errors and prevents variables from being used before they are set.
4. Use unique temporary filenames if creating any files or directories, using tools like `mktemp` to avoid potential conflicts or security issues.
5. Ensure that logging does not inadvertently print sensitive information.

2.0.772 `list_clusters`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 3f954969c054c0715b3cb4975cc16f60b59455fed57d76db287682a09738b747

2.0.773 Function overview

The `list_clusters()` function in Bash is used to list all clusters. It gathers cluster directories using the `_collect_cluster_dirs clusters` function call and stores the directories in a local array. The function also attempts to determine the active cluster name, ignoring any potential errors in case it is not set. If no clusters are found, the function alerts the user and returns an exit status of 0. Finally, it iterates over the array of clusters, and for each one, it checks if the cluster name matches the active cluster name and appropriately tags the active cluster in the output it echoes.

2.0.774 Technical description

- **name**: `list_clusters()`
- **description**: This bash function lists all clusters. It collects cluster directories, determines the active cluster (if any), and iterates over the cluster list to echo each one while highlighting the active cluster.
- **globals**:
  - `HPS_CLUSTER_CONFIG_BASE_DIR`: The base directory where cluster configurations are stored.
- **arguments**: None.
- **outputs**: Prints names of all clusters, marking the active one as '(Active)', if any.
- **returns**:
  - `0`: When no clusters are found in the `HPS_CLUSTER_CONFIG_BASE_DIR`.
  - Other values could be returned if inner functions (`_collect_cluster_dirs` and `get_active_cluster_name`) return them.
- **example usage**: Simply run the function without any arguments as `list_clusters`.

2.0.775 Quality and security recommendations

1. Error messages should be handled adequately. For instance, when no clusters are found, the program could inform the user on what steps to take.
2. The function could return unique exit codes for each type of failure to make debugging easier.
3. Consider sanitizing any user inputs or outputs, to prevent potential security risks.
4. The function should handle the possibility of not being able to collect cluster directories gracefully.
5. To achieve better code readability, consider adding more comments to explain complex code segments.
6. While the function does a good job managing local scope variables, careful attention must be paid to global variable usage. It could potentially cause conflicts with other parts of the program.

2.0.776 `list_local_iso`

Contained in `lib/functions.d/iso-functions.sh`

Function signature: 731918cdb2d287bfa33a94adacd15c3101a3688109bc456a782b3e30831ebc11

2.0.777 Function overview

The function `list_local_iso()` searches for ISO files in a specific directory. It uses three mandatory parameters: `cpu`, `mfr`, and `osname`, and an optional one `osver`. The ISO files it is looking for should fit the pattern `cpu-mfr-osname` or `cpu-mfr-osname-osver` if the version is specified. If the function identifies ISO files that match the given criteria, it will print the base names of these files. If not, it will print a warning message and return 1.

2.0.778 Technical description

- **Name:** list_local_iso
- **Description:** lists all local ISO files in specified directory which match the input pattern.
- **Globals:** None.
- **Arguments:**
  - `$1: cpu`: Description of the CPU architecture for which the ISO is intended, part of the ISO name pattern.
  - `$2: mfr`: Description of the ISO manufacturer, part of the ISO name pattern.
  - `$3: osname`: Description of the operating system, part of the ISO name pattern.
  - `$4: osver`: (Optional) Description of the operating system version, part of the ISO name pattern.
- **Outputs:** Prints the list of matching ISO files or a warning message if no matching files are found.
- **Returns:** If matching ISO files are found, it returns the base names of these files, or returns 1 if there are no matches.
- **Example usage:** `list_local_iso "x86" "intel" "ubuntu" "18.04"`

2.0.779 Quality and security recommendations

1. Always use full paths to directories and files to avoid potential ambiguity or misdirection.
2. Utilize error handling and error returns to manage unexpected input or conditions.
3. Avoid using shell options like `nullglob` as they may have unexpected side effects depending on the user's environment. Instead, ensure your code handles cases of no matches explicitly.

4. Guard against unexpected inputs especially if the function could be exposed to untrusted inputs at any time. Consider adding checks for valid inputs before running the function.

2.0.780 `_load_library_dir`

Contained in `lib/functions.d/node-libraries-init.sh`

Function signature: 6f6b91b5530fc47c78d9ed9b4fb917c6f93d8eb8da0fb48130ddb3c4ba74eeb8

2.0.781 Function overview

The `_load_library_dir` function is a Bash shell utility that loads Bash scripts from a specified directory. This might be used in a context where, for example, different scripts providing particular functionalities are organised into different directories, and we want to rather simply load all functionalities at once. The function takes as input a directory and a label, loads all the `.sh` files in the directory, and prints out the label and the names of loaded scripts. If the given directory does not exist or there are no `.sh` files, the function will not perform any operation and will return success. The loaded scripts are sorted in alphabetical order before they are loaded.

2.0.782 Technical description

- `name`: `_load_library_dir`
- `description`: Loops over all `.sh` files in the provided directory, sorts them in alphabetical order and loads them. It provides a debugging log message for each script it loads.
- `globals`: [`_LOADED_COUNT`: The amount of scripts loaded, increased each time a script is successfully loaded]
- `arguments`: [`$1: dir`, The directory from which the function will attempt to load .sh scripts, `$2: label`, A label which is printed before the filenames of the loaded scripts]
- `outputs`: The function outputs the label as a markdown header and the filenames of the loaded scripts prepended with `# Loading:`. Each script is also echoed.
- `returns`: The function will return 0 if the directory does not exist or has no `.sh` files or after it has successfully loaded the scripts.
- `example usage`: `_load_library_dir ~/scripts/ labelTest`

2.0.783 Quality and security recommendations

1. File permissions: Despite the fact that these scripts usually don't require elevated permissions, always be wary of who has write access to your files.

2. File validation: It's always beneficial to validate script files before executing them because they could contain surprising or even malicious content.
3. Error handling: Improve the script to handle errors more elegantly when failing to load a file.
4. Naming conventions: Use a more identifiable naming convention for scripts to be loaded.
5. Debugging: The function could provide more detailed logging information for easier debugging.
6. Handling symbolic links: Currently, the function does not handle symbolic links that might exist in the directory. This could be improved.

2.0.784 `make_timestamp`

Contained in `lib/functions.d/system-functions.sh`

Function signature: 7ea1fc9d3621ad0a04879323d75cd0a5f1aa2468bf98b9b3c83ff2b66dfa8e3d

2.0.785 Function Overview

The function `make_timestamp` is a simple Bash function that returns the current date and time in a specific format (Year-Month-Day Hour:Minute:Second UTC). The -u option makes sure that the command uses Coordinated Universal Time (UTC) instead of the local timezone.

2.0.786 Technical Description

- **Name:** `make_timestamp`
- **Description:** This function uses the `date` command with the `-u` option to get the current date and time in UTC, formatted as: Year-Month-Day Hour:Minute:Second UTC.
- **Globals:** None.
- **Arguments:** None.
- **Outputs:** Prints current date and time formatted as Year-Month-Day Hour:Minute:Second UTC.
- **Returns:** Nothing.
- **Example usage:**

```
$ make_timestamp
2023-01-01 00:00:00 UTC
```

2.0.787 Quality and Security Recommendations

1. This function has no user-input, hence it should be safe from security flaws that could result from unreliable user input.
2. Since the `date` command is a common Unix command, it should be available in most environments. However, in case the environment does not support the `date` command, appropriate error handling could be added.
3. For quality, consider adding checks if UTC timezone is required or if local timezone would suffice. If different timezones are needed, consider making the timezone an optional input to the function.
4. If the function will be used as part of larger scripts, consider returning the value instead of printing to allow better usage of this function.

2.0.788 `_matches_field`

Contained in `lib/functions.d/node-libraries-init.sh`

Function signature: 97146ad1f97962f042f48e084efe93fc1be505bcddd3ef38d36a995729d2c409

2.0.789 Function Overview

The function `_matches_field` is used to compare a pattern with a value. If there is a match, the function returns 0, otherwise it returns 1. The following aspects define a successful match: 1. If the pattern is a wildcard, represented as "*", it matches all so the function returns 0. 2. If the pattern exactly matches the value, the function also returns 0. 3. If the pattern contains multiple options, represented as comma-separated values, and any of these options match the value, the function returns 0.

2.0.790 Technical Description

- **Function Name**: `_matches_field`
- **Description**: This function compares a provided pattern with a given value. It returns success (0) if the pattern exactly matches the value or if the pattern is a wildcard ("*") or if any option in a comma-separated pattern matches the value. If no match is found, the function returns failure(1).
- **Globals**: None
- **Arguments**:
  - `$1` : The pattern. This can be a single string, wildcard ("*") or a comma-separated string.
  - `$2` : The value to be compared to the pattern.
- **Outputs**: This function does not print anything.
- **Returns**: 0 if there's a match, 1 otherwise.
- **Example Usage**:

```
# Match WildCard
_matches_field '*' 'any value'

# Exact match
_matches_field 'abc' 'abc'

# Comma-separated pattern match
_matches_field 'abc,def,ghi' 'def'
```

2.0.791 Quality and Security Recommendations

1. Always validate input before usage: Before invoking the function, ensure that both inputs, pattern and value, are well-formed and not malicious.
2. Be wary of possible globbing interference: If the pattern or value are entered via untrusted user input, they could contain unexpected globbing or regular expression symbols.
3. Regularly update the function to handle additional edge cases: Check for other types of patterns or data which this function might unexpectedly handle incorrectly.
4. If the function is not executing correctly, check for syntax errors in the call, such as forgetting to quote a string with spaces or special characters.
5. Avoid using excessive resources: If given large (especially infinite) inputs, this function could use a lot of CPU and memory. Prevent this by limiting the size of input.

2.0.792 `mount_distro_iso`

Contained in `lib/functions.d/iso-functions.sh`

Function signature: 37a39c7a765241cd6d873704bbe422852dd148c70c14f7a8fedaed1da85d834d

2.0.793 Function overview

This is a Linux shell function written in Bash that attempts to mount a specific operating system (OS) distribution ISO image file to a certain mount point. It validates if given OS identifier is valid and exists, checks if ISO file is present, readable, creates mount point if not already there, checks for already mounted or broken mounts, and tries mounting the ISO. Post, mounting it also confirms if mount succeeded and has any content. The function uses local and helper functions like 'os_config' and 'get_distro_base_path' in its operation.

2.0.794 Technical description

Name: `mount_distro_iso`

Description: This function is responsible for mounting an operating system (OS) distribution ISO file to a certain mount point, given the OS identifier.

Globals: None

Arguments: - `$1: The required Operating System (OS) identifier`

Outputs: Logs messages about the function's progress, errors, and the status of the mounting operation.

Returns: - Success (`0`) indicating successful mount operation or if it was already mounted. - Failure (`1`) indicating either invalid or missing OS identifier, ISO file not found or not readable, issues with creating mount point, mount failure, no content visible in mount point, etc.

Example Usage:

```
mount_distro_iso ubuntu
```

2.0.795 Quality and Security Recommendations

1. Always validate input data: The function could benefit from more thorough input validation. For instance, it could check if the OS identifier is of expected format or within the acceptable set of known identifiers, before proceeding with the rest of the operations.
2. Proper error handling: Currently the function returns 1 for various error scenarios. It would be useful to return distinct error codes for different issues to aid in troubleshooting.
3. ISO file validation: Beyond just checking if the ISO file is readable, the function could perform additional checks like verifying that it is indeed an ISO file (perhaps by checking file extension or file command output).
4. Security measures: The function should ensure that it doesn't leak sensitive information like paths or file names to standard output which could be logged or intercepted.
5. Use of recursive delete: The function should be cautious while using recursive actions like `ls -A` which could potentially cause high CPU/Memory usage if mount point contains a large number of files/directories.
6. Safeguard against path traversal attacks: It's important to ensure that the OS ID parameter cannot be manipulated to access unintended parts of the filesystem when forming paths. It would be prudent to add checks that prevent path traversal attacks.
7. Consider enabling auto-cleanup: For failure cases where mounting fails after the directory has been created, consider adding auto-cleanup logic that removes such stale directories after a certain period to prevent the accumulation of unused directories.

2.0.796 `netmask_to_cidr`

Contained in `lib/functions.d/network-functions.sh`

Function signature: 591afba929cc6e4793e6bae6dbae1c9f2b4f4429ff9102867ddd8a651f7f496d

2.0.797 Function overview

The `netmask_to_cidr` function is designed to accept a standard IP netmask as an argument and return the equivalent Classless Inter-Domain Routing (CIDR) notation. The CIDR notation is a compact representation of a network's IP address and its associated routing prefix. For example, the netmask `255.255.255.0` equals a CIDR of `24`.

2.0.798 Technical description

- **Name:** `netmask_to_cidr`
- **Description:** This function receives netmask as an argument and converts it into CIDR.
- **Globals:** N/A
- **Arguments:** `$1: netmask` (The standard IP netmask need to be converted into CIDR. E.g., 255.255.255.0)
- **Outputs:** The CIDR notation. (E.g., 24)
- **Returns:** 0 if it successfully echoes the CIDR. 1 if the provided netmask doesn't match any predefined value.
- **Example usage:**

```
$ cidr="$(netmask_to_cidr '255.255.255.0')"
echo $cidr  # Output: 24
```

2.0.799 Quality and security recommendations

1. Implement error checking to ensure the netmask provided is a legit netmask in the correct format.
2. Return error codes consistently so calling code can distinguish between different types of error situations.
3. Improve documentation, especially for the return values and what conditions lead to those return values.
4. Apply defensive programming principles such as checking that the function received exactly 1 argument, and return an error immediately if not.
5. Sanitize the input to prevent possible injection attacks. Although it's highly unlikely in this specific function, it's still a good habit to form in bash programming.

2.0.800 `network_calculate_subnet`

Contained in `lib/functions.d/network-functions.sh`

Function signature: 28bad9be5b146b636db38a5bc9383098acfa045e1d0212500adddc909c91161d

2.0.801 Function Overview

The `network_calculate_subnet` function in Bash is used for subnet calculation. It takes three arguments, namely a base IP, index, and CIDR notation, and then calculates the subnet. First, the function validates the inputs. If all inputs are valid, it extracts the first two parts (octets) of the base IP. The CIDR notation specifies how the remaining subnets should be calculated. The function also contains comment logs for visibility in case of unsupported CIDR notations.

2.0.802 Technical Description

- **Name:** `network_calculate_subnet`
- **Description:** Calculates the subnet based on given base IP, index, and CIDR.
- **Globals:** None
- **Arguments:**
  - `$1: base`: Base IP from which the subnetworks are calculated.
  - `$2: index`: Index used for calculations.
  - `$3: cidr`: CIDR (Classless Inter-Domain Routing) notation that specifies subnet's size.
- **Outputs:** Logs the calculated subnet or logs an error if CIDR is unsupported.
- **Returns:** `0` on successful calculation, `1` on error or if inputs are invalid.
- **Example usage:**

```
network_calculate_subnet 192.168.1 5 24
```

2.0.803 Quality and Security Recommendations

1. Using very specific subnets (like /25, /26, etc.) might lead to tabulation issues. Hence, it is recommended to use specific subnets only when necessary.
2. The function expects very specific inputs. Thus, it is advisable to add proper error messages conveying the required input in the case of a wrong input format.
3. The function can be modified to use arrays instead of three separate inputs for easier manipulation.
4. Make sure that the function is always used with proper sanitization of inputs to prevent possible injection attacks.

2.0.804 `network_ip_to_prefix`

Contained in `lib/functions.d/network-functions.sh`

Function signature: bd50dc3883248bb51b25dac61b99980ed8c1b451c68905c17e2185dc1e6f4f93

2.0.805 Function overview

`network_ip_to_prefix` is a bash function designed to parse a given IP address and output its network prefix (i.e., the first three octets of the IP address). This function relies purely on string manipulation without the use of external tools. If given an invalid argument or no argument at all, it will exit prematurely with a return code of 1.

2.0.806 Technical description

The technical specifications of `network_ip_to_prefix` function are as follows:

- **name**: network_ip_to_prefix
- **description**: Parses a given IP address string and echoes the network prefix (i.e., first three octets). Returns 1 and produces no output on invalid or missing input.
- **globals**: None
- **arguments**: [$1: A string consisting of four octets (IPv4 address) separated by periods]
- **outputs**: Echoes the network prefix of the input IP address
- **returns**: 0 if successful, 1 if unsuccessful due to invalid or missing input
- **example usage**: `network_ip_to_prefix 192.168.0.1 # outputs: 192.168.0`

2.0.807 Quality and security recommendations

1. The function can be refactored to handle both IPv4 and IPv6 addresses, increasing its utility.
2. Input validation could be improved by using a more rigorous method like regex, which can handle edge cases more effectively.
3. It is assumed that the input IP address is a well-formed, valid address. The function does not have any error checking or validation for an invalid IP address.
4. The function does not handle or sanitize any special characters in the input, potentially causing unintended behavior or security vulnerabilities.
5. The function should be designed to handle unexpected user input in a secure manner, instead of just assuming the input will always be as expected.

2.0.808 `network_subnet_to_network`

Contained in `lib/functions.d/network-functions.sh`

Function signature: 86f8adc9399482c2b4132b1a58360669317a1355449f2c1bdefdc599f5997e0f

2.0.809 Function overview

The function `network_subnet_to_network` is used to convert a network subnet into a network address. It does this by removing the CIDR suffix from the input subnet.

It takes the subnet as an argument, checks if it's not empty, removes the CIDR suffix and verifies if the result is not empty. If the subnet is empty or without a CIDR suffix, the function returns 1. Otherwise, it echoes the network address and returns 0.

2.0.810 Technical description

- **Name:** network_subnet_to_network
- **Description:** Converts a network subnet into a network address.
- **Globals:** None
- **Arguments:** [$1 - subnet: A string that represents a network subnet with a CIDR suffix.]
- **Outputs:** On success, it outputs the network address without the CIDR suffix. If the input subnet is empty or without a CIDR suffix, no output is produced.
- **Returns:** The function returns 0 on success, or 1 if the input subnet is empty or without a CIDR suffix.
- **Example usage:**

```
network_subnet_to_network "192.168.1.0/24"
```

2.0.811 Quality and security recommendations

1. Add more validation on the subnet input to ensure it properly formatted. This function assumes the provided subnet includes a CIDR suffix, but does not validate this.
2. Return error messages on stdout to inform the user why an operation failed. A no output situation might not be intuitive for the user.
3. Consider exit codes more expressive of the error or success nature, instead of just 0 and 1.
4. Use more distinctive variable names to avoid any potential overlaps with environment variables.

2.0.812 `network_subnet_to_prefix`

Contained in `lib/functions.d/network-functions.sh`

Function signature: 3259969db27545b0e8cbe25a3f952a034cb1e7575d9516c767866d50d9893fd0

2.0.813 Function overview

The function `network_subnet_to_prefix()` is a Bash function used to convert a network subnet to a prefix. It first retrieves the network address from the subnet provided and checks if the operation was successful and the network address is not empty. If everything is in order, it then proceeds to extract the prefix from the network address, again checking if the operation was successful and the prefix is not empty. The function

will output the prefix if it is retrieved successfully, else it will return 1 to indicate an error.

2.0.814 Technical description

- **Name:** `network_subnet_to_prefix()`
- **Description:** A bash function that converts a network subnet to its corresponding prefix.
- **Globals:** None.
- **Arguments:**
  - `$1: subnet` - The subnet to convert to a prefix.
- **Outputs:**
  - If successful: The network prefix.
  - If unsuccessful: Error status 1.
- **Returns:**
  - On success: 0
  - On error: 1
- **Example usage:**

```
subnet_prefix=$(network_subnet_to_prefix "192.168.1.0/24")
echo $subnet_prefix  # Outputs: 24
```

2.0.815 Quality and security recommendations

1. To make the function more robust, additional checks should be implemented to verify that the network subnet provided is in a consumable format, such as a string and falls within the expected ranges for subnets.
2. Handle exception for invalid input. For instance, instead of only checking if the returned prefix is empty, it could also check if it is a valid prefix number.
3. Ensure that the helper functions `network_subnet_to_network()` and `network_ip_to_prefix()` used during this computation also have appropriate validation mechanisms and securely handle potential errors.
4. Since the function does not currently involve any sensitive information processing, there are minimal direct security concerns. However, as a best practice, any future updates that might involve sensitive data need to have secure handling of such data.

2.0.816 `node_build_functions`

Contained in `lib/functions.d/node-libraries-init.sh`

Function signature: 807188926c4497aa2183999201297ce8ea50397cb17070307eb1ca632cd76846

2.0.817 Function Overview

The `node_build_functions` is a robust Bash function designed to handle the building of function bundles based on the specifications provided via its positional parameters. Its primary role is to load specified libraries (base, OS, type, profile, state) in hierarchical order, validate directories, and build an initialization sequence. It also includes utility functions for encoding and network, and includes some debug logs.

2.0.818 Technical Description

- Name: `node_build_functions`
- Description: it's used for function bundling which includes identification of OS, loading hierarchical libraries, utility functions and building an initialization sequence.
- Globals:
  - HPS_SYSTEM_BASE
- Arguments:
  - $1 (os_id): Operating System identifier
  - $2 (type): Type of the device/distinction feature
  - $3 (profile): Profile of the device or usage context
  - $4 (state): State of the device
  - $5 (rescue): Rescue option; boolean value
- Outputs: The function does not explicitly output a value but makes use of `echo` to show the result at several stages. The main output is the configuration bundle that gets built.
- Returns: The function returns `1` if the node manager directory doesn't exist. It returns `0` if the function successfully builds and creates the requisite functions bundle.
- Example Usage:

```
node_build_functions "x86_64:alpine:3.20.2" "server" "production"
 ↪  "running"
```

2.0.819 Quality and Security Recommendations

1. Set default values for the parameters. For instance, if the `rescue` command has a default value of `false`, it can prevent accidental triggering of the rescue process.
2. Leverage the set operation `-euo pipefail` to enforce bash script best practices such as error check on every line and prevention of uninitialized variables.
3. Integer comparison should be used when comparing integer values to prevent type mismatch issues.
4. Echo statements used for outputting information to the user should be replaced with proper logging mechanisms to ensure traceability.
5. Sanity checks should be in place to check if the necessary directories and files exist before proceeding with operation. This can prevent file or directory missing errors.

2.0.820 `normalise_mac`

Contained in `lib/functions.d/network-functions.sh`

Function signature: 597ad94d9559d604fae13c8fd8ef8bde5b6c19fee81c409691213ad492d3d6b3

2.0.821 Function Overview

The `normalise_mac` function in Bash is used to standardize the format of MAC addresses. It accepts a single MAC address as input, removes all common delimiters (such as "-" , ":" , "." or white space), converts it to lowercase, and validates the resulting output for being exactly 12 hexadecimal characters, which is the standard MAC address format.

2.0.822 Technical Description

- **name**: normalise_mac
- **description**: Normalizes the provided MAC address by removing common delimiters, converting to lowercase, and validating the MAC address format.
- **globals**: NA
- **arguments**: [$1: A string representing a MAC address. Input MAC address could be with any common delimiters like colon(:), hyphen(-), dot(.) or a whitespace.]
- **outputs**: The function outputs a standardized MAC address in lowercase and without delimiters if the input is valid. If the input doesn't match the standard MAC address format, it outputs an error message to stderr "[x] Invalid MAC address format: $1".
- **returns**: The function returns with a status of zero when successfully offering a normalized MAC address. It returns a status of 1 if the provided MAC address is invalid.
- **example usage**: `$ normalise_mac "01-23-45-67-89-ab"` `0123456789ab`

2.0.823 Quality and Security Recommendations

1. Be cautious while handling the argument and take steps to ensure that it does not carry any harmful inputs such as shell code injections.
2. Always use the function in a safe environment, taking into consideration input sanitation and privilege separation.
3. To enhance the function, consider adding more input validations, such as checking if the input is null or empty before processing.
4. Lastly, always inform the users of the function about the correct input to provide. Since the function expects an argument to be a MAC address, providing this in the function's help or documentation would help avoid confusion and unexpected output.

2.0.824 o_log

Contained in lib/functions.d/o_opensvc-task-functions.sh

Function signature: a128422c273b1df345526eab6502010e6b9cca98beb40ca6d2d3fe05aa5d3caa

2.0.825 Function Overview

The o_log function in Bash is built to manage logs. It is used for sending messages to the standard system logger. It takes up to three parameters. By default, the log priority is set to 'info' and the facility to 'user', with a predefined tag "opensvc". If the priority or facility arguments do not match predefined acceptable values, it defaults to 'info' and 'user' respectively while logging a warning about the input anomaly. A validation error occurs and the function halts if the message parameter is empty.

2.0.826 Technical Description

- Name: o_log
- Description: A bash function for logging messages to the standard system logger.
- Globals: None.
- Arguments:
  - $1: message - The log message
  - $2: priority - The priority of the message being logged. Acceptable values are 'err', 'warning', 'info', 'debug'. Defaults to 'info' if not specified or if specified value is invalid
  - $3: facility - The facility for the message. Potential values include 'user', 'local0' to 'local7', 'daemon', 'auth', 'syslog'. Defaults to 'user' if not specified or if specified value is invalid
- Outputs: Logs the given message with the specified priority and facility. If parameters are invalid, a warning is logged.
- Returns: The status code of the last executed logger command.
- Example Usage:

```
o_log "Application started" info daemon
```

2.0.827 Quality and Security Recommendations

1. As a quality improvement, incorporate a more detailed error handling system. Currently, when the function is called with an empty message, it only logs a simple error message. Instead, it could throw a detailed exception or return a specific error code.
2. Implement a stricter validation for priority and facility. Right now, the function only checks if the parameters match the expected values (and turns them into defaults if they don't). Consider integrating a proper error handling or feedback to users instead of silently changing the unwanted value to default.

3. Some input sanitation is suggested as a security measure. While Bash commands generally handle special characters, they might create unexpected behaviour with varying locales. So, a standardized message format is recommended.
4. Consider limiting the length of the log message to prevent any potential overflow issues. This can be integrated into the current validation section.
5. As part of best practices, always document the function with its expected inputs and outputs, which is already in place for this case.

2.0.828 o_node_label_add

Contained in `lib/functions.d/o_node-label-functions.sh`

Function signature: 7662084d4bf2ccba6e23772abd53850712c663ff7956e960844c09c8b7fcd8b8

2.0.829 Function Overview

The `o_node_label_add` function in Bash is designed to add a specific label to one or multiple nodes. The function takes three arguments, namely the nodes to be labeled, a key for the label, and a value for the label. It validates the parameters, ensures the nodes exist, and then iterates through all the nodes to apply the label. This function uses logs to display information or errors, and has built-in error handling to return different statuses depending on the outcome.

2.0.830 Technical Description

- **Name**: `o_node_label_add`
- **Description**: This function adds a specific label (defined by key and value) to one or more nodes. If the nodes parameter is “all”, it will apply the specified label to all existing nodes. It validates all parameters and handles errors by logging and returning status codes.
- **Globals**: No global variables are modified by this function.
- **Arguments**:
  - $1: nodes - The nodes to which the label should be added. Can be an array of node names or “all” to apply label to all nodes.
  - $2: label_key - The key of the label to be added.
  - $3: label_value - The value of the label to be added.
- **Outputs**: This function logs info and error messages during its execution.
- **Returns**:
  - 0: All operations completed successfully.
  - 1: Function was provided with invalid arguments or failed to get node list.
  - 2: Failed to add label to one or more nodes.
- **Example Usage**: `o_node_label_add "node1 node2" "env" "prod"`

2.0.831 Quality and Security Recommendations

1. Instead of comparing arguments to an empty string within the function, consider allowing Bash to enforce arguments during invocation by setting them as required.
2. For greater traceability and debugging, add more logging steps within the function, indicating the process along with the nodes to which the labels are being added.
3. To bypass arbitrary command execution, sanitize all user inputs provided as arguments.
4. Consider using double brackets [[ ]] for comparison operations as they are a safer and more powerful way of checking conditions in Bash.
5. Ensure om CLI tool is correctly installed and configured as this function is dependent on it.

2.0.832 `o_node_label_exists`

Contained in `lib/functions.d/o_node-label-functions.sh`

Function signature: bf7eb0ec198341162d77fbfc57b8af2754d485e6cb2e9a77c980a30edb2eb1ed

2.0.833 Function Overview

The function `o_node_label_exists()` is a Bash function that checks if a given node label exists in a system. By supplying it with a node, a label key, and optionally a label value, it determines whether the label configuration contains the specified key-value pair. After validating the parameters, it retrieves the node's label configuration and verifies if the supplied label key exists. If a label value is provided, the function also checks if it matches the existing value. It returns different status codes based on the results, making it an essential tool for instance configurations, especially in the context of node labelling in a cluster management.

2.0.834 Technical Description

Name: `o_node_label_exists()`

Description: This function checks whether a given node label key (and optional value) exist. It takes three arguments, the name of the node, the label key, and optional label value. After getting configuration of node labels and checking for the key (and value if provided), it returns specific statuses depending on the result.

Globals: None

Arguments: - `$1: node` - The node name to check label in. - `$2: label_key` - The label key to search for. - `$3: label_value` - An optional value of the label.

Outputs: Diagnostic messages are redirected to the error stream (`stderr`).

Returns: - `0` if the key (and value if provided) exists - `1` if either node or label key is not provided - 2 if the label key (or value if provided) does not exist

Example Usage:

```
o_node_label_exists "node1" "label1" "value1"
```

2.0.835 Quality and Security Recommendations

1. Always remember to provide a description for each return status code. This can help other developers to easily understand what each status code stands for.
2. The `o_node_label_exists` function does not check whether the input parameters are valid or safe. It's recommended to add validation procedures for input and also sanitize the input to prevent any form of security breach e.g. script injection.
3. Ensure that all global variables used by the function are clearly defined and initialized. Although this function does not use any global variables, it is a good practice to keep track of them. This would eliminate any chance of accidentally overwriting valuable data or exposing sensitive information.
4. Avoid suppressing output with `/dev/null` unless necessary. Output can often be a useful tool for debugging and auditing.
5. Always secure your bash scripts using the right file permissions. Scripts that can alter the system configuration should only be editable by trusted users.

2.0.836 `o_node_label_list`

Contained in `lib/functions.d/o_node-label-functions.sh`

Function signature: a0af8a5747a5ad1f290dc1c50ec32ab7738495c9d34327fd7c14a1a7f806e6c8

2.0.837 Function Overview

The Bash function `o_node_label_list()` is used to query nodes based on their labels. The function takes up to three arguments - a required label expression in the key=value format, an optional logic operator (either "and" or "or"), and a flag indicating whether the function should operate in quiet mode (i.e., suppress logging). Based on the provided logic operator, it queries the OpenSVC configuration database and retrieves a list of nodes that match the provided label expression. If the logic operator is 'or', the function passes all labels directly to the OpenSVC 'om node ls' command, but if the operator is 'and', it queries each label individually and retrieves an intersection of the results. Once the function has retrieved a list of nodes, it logs the result and outputs the node list to standard output.

2.0.838 Technical Description

- Name: `o_node_label_list`
- Description: A bash function that queries nodes based on their labels and, depending on the logic operator used ('or' or 'and'), returns a list of nodes matching the given label expressions.

- Globals: `quiet`: a flag that suppresses logging when set to true.
- Arguments: `$1: label_expression`: key=value format string; `$2: logic`: string that accepts either 'or' or 'and' as values; `$3: quiet`: boolean value indicating whether logging should be suppressed.
- Outputs: A list of nodes matching the input label_expression, printed to stdout.
- Returns: 0 on successful execution, 1 if any of the input arguments are invalid or absent, 2 if the nodes query process in the OpenSVC command fails.
- Example Usage: `o_node_label_list "tier=prod" "and" "false"`

2.0.839 Quality and Security Recommendations

1. Include a validation step to ensure the function is invoked with valid input parameters. This way, potential errors are caught early, and unexpected behaviour prevented right from the start.
2. Avoid making the script run in interactive mode without requiring user consent. Running in quiet mode can suppress essential feedback, which may go unnoticed and lead to data discrepancy.
3. Make sure command well-defined, and it can handle exceptions when it fails to retrieve node labels from the OpenSVC command.
4. Include error-handling mechanism when the `om node ls` command fails. This will prevent the instance where the function quietly fails without notifying the user.
5. Validate that label expression is of the correct format before making requests.

2.0.840 `o_node_label_remove`

Contained in `lib/functions.d/o_node-label-functions.sh`

Function signature: 96293dde745044d68fd3015a9a6e1be7ff6061c2e50192f0891d8258f6c0add7

2.0.841 Function Overview

The function `o_node_label_remove()` is designed to remove a specified label from a node or a list of nodes. The function takes two arguments - `nodes` (a lists of nodes) and `label_key` (the key for the label to be removed). The function validates these arguments, logs an error message if they are not provided, and stops execution. If 'all' is passed as nodes, the function expands this to all of the nodes and continues the process.

2.0.842 Technical Description

- **name:** `o_node_label_remove()`
- **description:** Removes a specified label from a node or a list of nodes.
- **globals:** None
- **arguments:**

  - `$1 (nodes):` A string representing the target node(s)
  - `$2 (label_key):` The key of the label which is to be removed
- **returns:**
  - `0:` If the removal of labels from all nodes is successful.
  - `1:` If either one or both of parameters are not given.
  - `2:` If failed to remove label from atleast one node.
- **outputs:** Log messages for each removal attempt (success or failure) and summary.
- **example usage:**

```
o_node_label_remove node1 label1
```

2.0.843 Quality and Security Recommendations

1. Implementation of more robust error handling and parameter checking could increase the robustness of the function.
2. Adhere to best-practices for logging, such as including timestamps and the level of severity for each message.
3. Checking the return status of each command execution could help in troubleshooting and ensure fewer unexpected behaviours.
4. Consider escaping or sanitizing the input to prevent potential command injection vulnerabilities.
5. If the list of nodes can be very long, consider implementing the function to work with batches to reduce resource consumption.

2.0.844 `o_node_test_logger_ips`

Contained in `lib/functions.d/o_opensvc-task-functions.sh`

Function signature: 3271a1b737f6ea9281c317b031310a02e4a6a8d0e4f3ea9b4a8e1fbd2f2b3f15

2.0.845 Function Overview

The `o_node_test_logger_ips` is a bash function which is primarily used for logging. It logs a variety of information like the host name, timestamp, process ID, and a custom message to the log file. It is designed to be used with the `hps_log` utility on an IPS host, and it utilizes a local custom message which if not provided, defaults to “No custom message”.

2.0.846 Technical Description

- **Name**: `o_node_test_logger_ips`
- **Description**: This function logs several information like the current host name (via the `hostname` command), timestamp, process ID, and a custom message. It calls the `hps_log` utility to perform the logging.

- **Globals**: None
- **Arguments**:
  - `$1: Custom message`. This argument if provided will be used as the custom message in the logs. If not provided, the function defaults this to "No custom message".
- **Outputs**: This function does not produce a standard output as its main function is to log messages to a specified log file through `hps_log`.
- **Returns**: It returns 0 implying that it completes successfully without any error.
- **Example usage**: `bash o_node_test_logger_ips "Custom message for test log"`

2.0.847 Quality and Security Recommendations

1. It is necessary to ensure that the `hps_log` utility, used within the function, does not have any vulnerabilities which could be potential paths for attacks.
2. Review the requirement if the logging of process ID is necessary to avoid any security risks as it can provide crucial information to an attacker.
3. Instead of using `$(hostname)`, the `uname -n` command can be used as a safer alternative.
4. Validate the input `$1` to prevent command injection attack. Unvalidated input could allow an attacker to input a string which can execute unintended commands.
5. Ensure proper access control on the logs being written by the function to prevent unauthorized access.
6. Use more descriptive names for function, and parameters to enhance the readability of the code. Improving code readability makes it easier to maintain, thus reducing potential coding errors and associated security risks.

2.0.848 `_opensvc_foreground_wrapper`

Contained in `lib/functions.d/opensvc-function-helpers.sh.sh`

Function signature: fe503eb64a54d99b33e3d9a582d5c6ae42ff10189747941a5380e5e7c3f2d848

2.0.849 Function Overview

The function `_opensvc_foreground_wrapper()` is used to manage daemon processes in the foreground for the OpenSVC project. This allows for external supervisor tools to manage these daemons if required. The function also sets a logging directory, defaulting to `/srv/hps-system/log` if no other path is provided.

2.0.850 Technical Description

- name: `_opensvc_foreground_wrapper`

- description: This function manages daemon processes for the OpenSVC project in a supervisable foreground state and sets up a logging directory.
- globals:
  - HPS_LOG_DIR: This global variable holds the value of the desired log directory. If not provided, the function defaults to /srv/hps-system/log.
- arguments: None
- outputs: The function exec command routes stdout and stderr of the daemon process to the logger command which logs them in the set log directory with priority local0.info.
- returns: Nothing since the exec command replaces the shell without creating a new process.
- example usage: _opensvc_foreground_wrapper

2.0.851 Quality and Security Recommendations

1. The function uses the exec command to replace the current shell with the daemon process. Although this is good as it doesn't create a new process, it also means that if the exec command fails, then the shell is exited, possibly leaving the system state in an undesirable manner. It's recommended to handle such cases by checking if the command executed successfully.
2. If the HPS_LOG_DIR variable is not sanitized before usage, it might result in path traversal vulnerabilities. Always sanitize user inputs before using them.
3. It would be beneficial to have error reporting or logging in place for when the default LOGDIR is used, indicating that the HPS_LOG_DIR was not set correctly.
4. The logger command's log priorities should be managed dynamically for efficient logging of different levels of system information.
5. This function does not take any input parameters. If future versions might require this, consider using error checking or validation for input parameters.

2.0.852 _opensvc_foreground_wrapper

Contained in lib/functions.d/opensvc-function-helpers.sh.sh

Function signature: fe503eb64a54d99b33e3d9a582d5c6ae42ff10189747941a5380e5e7c3f2d848

2.0.853 Function Overview

The Bash function _opensvc_foreground_wrapper is designed to run the v3 daemon in the foreground, enabling management through a supervisor if required. It first determines the log directory by checking if the HPS_LOG_DIR global variable has been set; if not, it defaults to /srv/hps-system/log. It then executes the om daemon run command, piping the output to the logger command with a tag of om and a priority of local0.info.

2.0.854 Technical Description

- **Name**: `_opensvc_foreground_wrapper`
- **Description**: This function runs the v3 daemon in the foreground, allowing a supervisor to manage it.
- **Globals**: [`HPS_LOG_DIR`: The preferred log directory. If not set, it defaults to `/srv/hps-system/log`]
- **Arguments**: None
- **Outputs**: Logs from the v3 daemon and outputs of `logger` command
- **Returns**: The execution status of `om daemon run` command
- **Example usage**: Call the function using `source` or `.` (dot) like so: `source _opensvc_foreground_wrapper` or `._opensvc_foreground_wrapper`

2.0.855 Quality and Security Recommendations

1. Add error handling: The function currently doesn't handle any errors that can occur during execution. Enhance it with error handling to make it more robust.
2. Validate the `HPS_LOG_DIR` variable: The function uses the `HPS_LOG_DIR` variable without validating its content. Ensure it exists and is a valid directory.
3. Protect against command injection: Use safeguards to prevent any potential command injection vulnerabilities when using the `exec` function.
4. Document the function: Include comments within the function, explaining what each line or block of code does for better maintainability.
5. Use Least Privilege Principle: Ensure that the scripts that run this function have only the necessary permissions required to perform their intended tasks. This will minimize the potential damage from errors or security breaches.

2.0.856 `os_config_by_arch_and_type`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: 9f28855dbbcdc6606b83f708e615a1ececc2115d7c943654bc0a9add73ec95cf

2.0.857 Function overview

`os_config_by_arch_and_type` is a function that takes two parameters, required architecture (`req_arch`) and host type (`host_type`), to identify and echo the right OS configuration. It loops through a list of OS configuration IDs (the list is fetched using a call to another function `os_config_list`). Then it extracts the architecture from the ID, checks the architecture from configuration setup, and fetches the host types belonging to that OS configuration. If the identified architecture matches the required one, and the host types contain the required host type, it echoes the OS ID and changes the `found` flag to 1, signifying that the required OS configuration has been found. The truthiness of the `found` flag is returned as the function output.

2.0.858 Technical description

- Name:
`os_config_by_arch_and_type`

- Description:
This function is used to find the matching OS configuration by given architecture and host type.

- Globals:
None

- Arguments:
* `$1`: The required architecture (`req_arch`). * `$2`: The required host type (`host_type`).

- Outputs:
Prints the OS configuration ID that matches the required architecture and host type.

- Returns:
Returns 0 if a matching OS configuration is found, else returns 1.

- Example usage:

```
os_config_by_arch_and_type "x86_64" "server"
```

This will echo the OS configuration ID for architecture `x86_64` and host type `server` if it exists.

2.0.859 Quality and security recommendations

1. Add input validation: Check if input arguments are empty or not, or if they are in the expected format, to reduce the risk of incorrect inputs causing unexpected behavior.
2. Error handling: Include better error handling in case any of the functions (`os_config_list`, `os_config`) fail, which may disrupt the whole process.
3. Code readability: The function currently uses complex logic with nested if conditions and checks. Simplifying the logic where possible would aid readability.
4. Use explicit variable declaration: `local` is used for scoping within the function, which is good for minimizing unexpected side effects. But it could be more explicit about what data those variables are expected to contain (e.g., strings, integers, arrays).
5. Commenting: Provide more descriptive comments for understanding the business logic and improving maintainability.

2.0.860 `os_config_by_type`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: b81425e5107d65d999a2e7ed819a5554f07fa1d8d243414c11f28b5c8124956d

2.0.861 Function Overview

`os_config_by_type()` is a Bash function designed to search for a particular type of operating system configuration from a list. It requires a host type input to identify which operating system configuration type it should look for in the list. The function will continue to search through the list until it finds a match or exhausts all possibilities. Once a match is found, it returns the corresponding `os_id` to the user.

2.0.862 Technical Description

name: `os_config_by_type`

description: This function iterates through a list of operating system configurations, searching for a specific host type given as an argument. Once a match is found, it outputs the `os_id` and breaks off the loop.

globals: None

arguments: - `$1: host_type`. The type of host operating system configuration to search for in the list.

outputs: Prints the `os_id` for the matching configuration. If no match is found, no output.

returns: - `0`: if a match for the host_type is found. - `1`: if no match is found.

example usage:

```
os_config_by_type "debian"
```

2.0.863 Quality and Security Recommendations

1. Always validate input: The function should verify the `host_type` argument to ensure it is in an expected format before processing.
2. Error handling: Include error handling to address potential issues like if the `os_config_list` does not exist or if the `os_config` command fails.
3. Documentation: Provide more detail in comments around the logic of the function, especially the regular expression used in the matching process.
4. Security: Be aware of potential security vulnerabilities, such as command injection vulnerabilities, that could be exploited via the `host_type` argument.
5. Return meaningful error messages: Instead of just returning `1` when no match is found, consider returning a helpful message that states no host_type match was found.

2.0.864 `os_config_exists`

Contained in `lib/functions.d/os-functions.sh`

Function signature: a976e048233b022a1bd8b4182c5a1db97b6842b6a504f3531bbfda65187387fa

2.0.865 Function Overview

The `os_config_exists()` function is a bash function that checks if a particular operating system configuration exists in a predefined configuration file. The function takes as input an `os_id` that specifies the operating system. It then locates the configuration file path using the `_get_os_conf_path()` function. It checks if the configuration file exists and if it does, uses `grep` to see if the `os_id` is present within the configuration file. It returns the status of this operation.

2.0.866 Technical Description

Name: `os_config_exists()`

Description: This function checks if the given operating system configuration exists in a pre-defined configuration file. It utilizes a helper function `_get_os_conf_path()` to get the file path of the configuration file. If the file exists, it searches for the OS ID within the file.

Globals: - `os_conf: This is a variable storing the absolute file path of the operating system's configuration file.`

Arguments: - `$1: This argument accepts a string which represents the ID of the operating system whose configuration is to be checked.`

Outputs: - The function will output the status of the grep operation for the operating system ID in the configuration file.

Returns: - `0 if the OS configuration exists; 1 otherwise`

Example Usage:

To check if an operating system with ID 'ubuntu20' exists in our configuration, you would use:

```
os_config_exists 'ubuntu20'
```

2.0.867 Quality and Security Recommendations

1. Add additional checks for the existence of OS configuration file obtained from `_get_os_conf_path()`.
2. Validate the input argument `os_id`, i.e., check if the provided `os_id` is a valid string and matches your expected pattern and format.
3. Make use of error messages to provide more details about the failure, for example, when the configuration file doesn't exist or `os_id` is not in the right format.
4. Secure the file containing the configuration data to prevent unauthorised modification.
5. If not required, avoid global variables which can be altered anywhere in the bash script causing unpredictable results.

2.0.868 os_config_get_all

Contained in lib/functions.d/os-function-helpers.sh

Function signature: 76fce9ff8c97427dc901265bd06bb6cd7a19f2cb0f2d318ee8c06439d6b9026f

2.0.869 Function overview

The os_config_get_all() function is a bash function designed to read and output all key-value pairs present in a configuration file related to a specific operating system. The function takes an operating system ID as its argument. It then loops through a configuration file, identifies the section associated with the provided OS ID, and outputs the configuration details for that OS in terms of key-value pairs.

2.0.870 Technical description

- **Name:** os_config_get_all()
- **Description:** This function reads a configuration file and outputs all key-value pairs related to a specific Operating System. It is used for retrieving all settings related to an OS from a configuration file.
- **Globals:** [os_conf: The path to the configuration file]
- **Arguments:**
  - $1: The Operating System ID for which the configurations should be returned
  - $2: Not used in this function
- **Outputs:** The function will output all key=value pairs for the provided OS ID from the configuration file.
- **Returns:** Return code 1 if the operating system configuration section is not present or the file does not exist, 0 if the section is found and the key-value pairs are successfully read.
- **Example usage:** ~~~ os_config_get_all “ubuntu” ~~~

2.0.871 Quality and security recommendations

1. Add continuously updating system log mechanism to monitor unexpected behavior.
2. Implement additional error checking, beyond just whether the file exists and whether the provided OS has a section in the configuration file. For example, checks for whether the file is readable, and whether the OS ID is in a valid format.
3. Secure the configuration file with the appropriate permissions to ensure it is not accidentally deleted or maliciously modified.
4. Ensure that there is unit testing in place for this function to catch any potential bugs or points of failure.
5. Avoid hardcoding path names and file names. Make use of environment variables and function arguments to increase reusability.

6. Consider encrypting key-value pairs if the data in the configuration file is sensitive.

2.0.872 `os_config_get`

Contained in `lib/functions.d/os-functions.sh`

Function signature: 097d452d312223a9b541a8cfd4606b74c649545a82a077f1eaf0a70554a8eb48

2.0.873 Function overview

The `os_config_get` function is a Bash utility that retrieves a key's value from an Operating System's configuration. It takes the OS ID and a key as its parameters. The function checks a pre-defined OS configuration file for the requested key under the specified OS's section and returns the associated value, if it exists.

2.0.874 Technical description

- **name**: `os_config_get`
- **description**: This function accepts an OS identifier and a key and returns the corresponding value from the OS configuration file. If the given key is not found under the specified OS's section in the configuration file, the function returns a failure state.
- **globals**: `_get_os_conf_path`: An underlying function that evidently returns the OS configuration file path.
- **arguments**:
  - `$1`: OS ID, the identifier of the Operating System.
  - `$2`: Key, the configuration key for which the value needs to be fetched.
- **outputs**: The function echoes the value associated with the specified key if found, where it can be caught into a variable by the calling function. If not found, there will be no STDOUT output.
- **returns**: The function returns zero (0) when the key is successfully found and its value has been echoed. If the key isn't found or there is an issue with the configuration file, it returns non-zero (1).
- **example usage**:

```
os_conf_value=$(os_config_get $os_id "config-key")
if [[ $? -ne 0 ]]; then
  echo "Unable to fetch configuration value"
else
  echo "Fetched value: ${os_conf_value}"
fi
```

2.0.875 Quality and security recommendations

1. Add more explanatory comments to improve readability and maintainability of the function.

2. Error messages should be sent to STDERR instead of STDOUT.
3. Add additional checks to validate the inputs (OS ID and key).
4. Instead of returning only 1 for different kinds of failures, unique exit status codes for each type of error can be introduced for better debugging.
5. The function implicitly depends on the `_get_os_conf_path` function to get the configuration file. Making this dependency explicit would improve readability and function portability.

2.0.876 `os_config_latest_minor`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: b8c3da218094f10a34d4ac207b0b2fc26189d7e311246d27e6e7befa5da1f2d1

2.0.877 Function overview

The `os_config_latest_minor()` function scans through the list of operating system configurations and returns the identifier of the one with the latest minor version. It allows filtering these operations systems based on their status. The function uses a version comparison utility to accurately determine the latest version.

2.0.878 Technical description

- **Name**: `os_config_latest_minor()`
- **Description**: This function iterates over available Operating Systems configs, filtering them according to a status if provided, and gathering full version data. It uses basic comparison to seek out the most recent minor version, keeping track of the latest version and its identifier.
- **Globals**: None
- **Arguments**:
  - `$1`: a pattern to filter the list of OS configurations.
  - `$2`: an optional status filter. If provided, it will be used to filter OS configurations by status.
- **Outputs**: Prints the identifier of the configuration with the latest minor version on standard output.
- **Returns**: Returns nothing.
- **Example usage**: `os_config_latest_minor my_pattern my_filter`

2.0.879 Quality and security recommendations

1. It would be wise to add input validation to ensure the `pattern` and `status_filter` are valid, and possibly limiting the length of these parameters to prevent potential security vulnerabilities.

2. This function may fail silently due to redirection of standard error (`2>/dev/null`). For better error handling, consider logging errors into a designated error file.
3. The function relies on `os_config_list` and `os_config` commands without any kind of checking whether these commands exist and perform as expected. It is advisable to add some command availability checks.
4. As the function can potentially take a long to run if there are many OS configurations, implementing some sort of progress reporting could be beneficial.

2.0.880 `os_config_list`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: 9979b88d45894823c4cc985b816422d7701b4ef3bf2444f53bb9e0efe6b95bc7

2.0.881 Function overview

This bash function `os_config_list` is designed to read the operating system configuration file and list all the configuration sections therein. It first acquires the path of the configuration file by calling another function (_get_os_conf_path), and then uses grep and sed commands to extract and clean up the section names in the configuration file.

2.0.882 Technical description

Name: os_config_list

Description: The os_config_list function lists all the sections within the OS configuration file, returning 1 if the file doesn't exist. It uses a more robust pattern to handle whitespace and colons within section names.

Globals: None

Arguments: None

Outputs: This function outputs a list of the sections (names) in the configuration file, delimited by newline characters. The output is sent to stdout.

Returns: The function will return 1 if the configuration file is not found, otherwise no specific return value is given.

Example usage:

```
$ os_config_list
section1
section2
section3
...
```

2.0.883 Quality and security recommendations

1. Due to the function's dependency on a separate function to obtain the configuration file path, it is recommended to ensure that the function `_get_os_conf_path` is robust and secure, and that it returns a valid file path.
2. To improve maintainability, consider documenting the expected format or structure of the configuration file. This will also help reduce potential errors, especially if other functions depend on this specific format.
3. Ensure that there are proper checks or exceptions in place when the configuration file is not present, when incorrect configuration file path is provided, or if there are any permissions issues.
4. Regularly monitor and log function usage. This will help in error tracking, problem diagnosis, and maintaining data integrity. Unauthorized attempts at function usage can also be caught this way.

2.0.884 `os_config`

Contained in `lib/functions.d/os-functions.sh`

Function signature: 2c1cebb97829a2157aeb56ed995ebd8e235013520bde30442ff7f5d174082819

2.0.885 Function overview

The `os_config()` function is used to manage operating system configurations. It takes in four arguments - `os_id`, `operation`, `key`, and `value`. It's capable of performing four operations: `get`, `set`, `exists` and `undefine`.

2.0.886 Technical description

- **Name:** os_config
- **Description:** This function manages operations on OS configurations. It can perform get, set, exists and undefine operations depending on the arguments provided.
- **Globals:** None
- **Arguments:**
  - `$1(os_id)`: ID of the operating system configuration
  - `$2(operation)`: Operation to perform. It can be either get, set, exists, or undefine
  - `$3(key)`: Key to perform operation on. They are optional for the `exists` and `undefine` operations
  - `$4(value)`: Value to set. This is only used in the `set` operation and is optional

- **Outputs:** The output depends on the operation performed:
  - For `get`, it prints the configuration value of the given key
  - For `set`, it does not output anything
  - For `exists`, it checks if configuration is defined
  - For `undefine`, it removes a key or section in the configuration
  - If an invalid operation is passed. it prints an error message.
- **Returns:** For invalid operations, it returns 1.
- **Example usage:** `os_config 123 get config_key`

2.0.887 Quality and security recommendations

1. Put more detailed checks for the input parameters, especially for `os_id` to ensure valid inputs
2. Avoid printing error messages directly in functions and instead throw exceptions or let the calling function handle the errors. This would provide more robust error handling.
3. Implement a proper logging mechanism instead of using simple echo statements.
4. Always sanitize and validate input especially when it's coming from an insecure source.
5. For the `exists` and `undefine` operations, add an explicit check for the key.

2.0.888 `os_config_select`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: f6ce9220f6052b6d82b8709ea8dd24da762af4f9d5f10c375090e8e00268fdfd

2.0.889 Function overview

The `os_config_select()` function operates within an OS configuration system. Given an architecture (`arch`), host type (`host_type`), and optionally a version preference (`version_pref`), it will attempt to find an ideal operating system configuration. It first attempts to directly match the given parameters of architecture and version preference, and the output will be linked with the host type. Failing that, it attempts to find the latest minor version of the preferred version for the same architecture. If none of these options work, the function will then attempt to find any production OS that matches the given architecture and host type. If successful at any of these steps, it outputs the chosen OS ID. Otherwise, it returns 1.

2.0.890 Technical description

- **Name:** `os_config_select()`
- **Description:** Tries to find an OS configuration that matches the passed parameters.

- **Globals:** Not applicable
- **Arguments:**
  - `$1: arch` - The architecture of the OS
  - `$2: host_type` - The host type of the OS
  - `$3: version_pref` - The version preference for the OS (optional)
- **Outputs:**
  - Exact or latest minor OS config ID if an OS config matches given parameters
  - Any production OS config ID if none of the above applies
- **Returns:**
  - `0`: if a suitable OS config is found
  - `1`: if no suitable OS config is found
- **Example usage:** `os_config_select x86_64 test_host 3.10`

2.0.891 Quality and security recommendations

1. Make sure you validate the input arguments to ensure they are in the expected format and within the expected range.
2. Since the function calls other functions like `os_config`, `os_config_latest_minor`, and `os_config_by_arch_and_type`, ensure their implementation is also secure and error-proof.
3. Handle exceptions properly, especially when the function relies on the output from other functions.
4. Avoid using global variables as they can be modified outside the function, causing hard-to-track bugs.
5. Add more detailed comments to increase understandability and code maintainability.
6. Configure your Bash script to stop executing if any command exits with a non-zero status. This can be achieved by adding the line `set -e` at the top of your Bash script.

2.0.892 `os_config_set`

Contained in `lib/functions.d/os-functions.sh`

Function signature: c4746ca6605cff53d6621b31b6cd996758390233161c17381459ddfcdb47c0a5

2.0.893 Function Overview

The `os_config_set` function is a shell function developed for bash scripts that deal with configuration files on a Unix-like operating system. It takes in three arguments: `os_id`, `key`, and `value`. It checks a specified configuration file for a particular section (defined by `os_id`), then updates it (or creates it) with a specified `key-value` pair. If the configuration file is empty or if the specified section does not exist, the function creates a new section.

2.0.894 Technical Description

- **name:** `os_config_set`
- **description:** This function updates or creates a section in a configuration file with the given `os_id` as the section name. Inside this section, it either updates an existing `key` with a new `value` or adds the new `key-value` pair if the `key` does not exist.
- **globals:** None
- **arguments:** `$1(os_id)`: The identifier of the OS configuration section, `$2(key)`: The configuration key to set, `$3(value)`: The value to set for the key.
- **outputs:** Rewrites a configuration file by updating or adding a new `key-value` pair in a specified section.
- **returns:** It will always return `0` indicating successful execution.
- **example usage:**

```
os_config_set "ubuntu18" "hostname" "my-new-host"
```

2.0.895 Quality and Security Recommendations

1. To avoid errors or unexpected behavior, check if the `os_id`, `key`, and `value` variables are not empty before proceeding with the rest of the function.
2. Instead of directly writing to the configuration file, consider creating a backup of the original file and then apply changes to the backup. This way, if something goes wrong during the operation, the user can revert to the original configuration.
3. It may be beneficial to include error messages or logs when a section does not exist in the configuration file, or when the desired key to be updated is not found.
4. Check all output of external commands using `$?` to make sure the command was successful. For instance, check the output of the `mv` command used towards the end of the function.

2.0.896 `os_config_summary`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: 37b0e6d65277922f2e599a5991f70fead11c1a1b295c0f577a17133125a8561a

2.0.897 Function Overview

The function `os_config_summary()` retrieves configuration information about various operating systems and presents them in a neatly organized manner. It groups the information by architecture and echoes out each configuration with details like host types, OS with version, status, update details, etc.

2.0.898 Technical Description

- **Name**: `os_config_summary`

- **Description**: The function sources a configuration file (retrieved from a separate Bash function `_get_os_conf_path`). It ensures the file exists, processes the file, categorizes data by architecture, and then prints each configuration's specifics.
- **Globals**: None
- **Arguments**: None
- **Outputs**: Prints an overview of available OS configurations, grouped by architecture and including details such as host types, OS name and version, status, last update, and notes.
- **Returns**: 0 if the function runs successfully, 1 if the OS configuration file does not exist.
- **Example Usage**:

```
os_config_summary
```

2.0.899 Quality and Security Recommendations

Below are some suggestions for security and quality improvement:

1. Equipped with more specific error handling, and error messages that guide the user to troubleshoot issues themselves.
2. Avoid using `echo` to print variable data, it can lead to issues with unusual input. Use `printf` instead.
3. Ensure variable expansions are placed within double quotes to prevent word splitting and pathname expansion.
4. Consider adding validation for data retrieved from `os_config`.
5. Dedicated function for handling the printing of the configuration information could be a good idea, so it can be reused in other parts of the program.
6. `os_config_list` and `os_config` are two separate functions used in this function. It would be good to check for their existence before invoking them.
7. Rather than using `echo "N/A"` for missing information, you might consider using a confirmation that the value was not found, offering more useful feedback to the user.

2.0.900 `os_config_undefine_key`

Contained in `lib/functions.d/os-functions.sh`

Function signature: 02ee32e648387b7a410b85156547b77c4b5c672f5431af30b93b99d2fbfe1cc9

2.0.901 Function Overview

The function `os_config_undefine_key()` is used to remove a specific key from a specific OS configuration setting in a configuration file. It achieves this by reading each line in the file and checking for a section header that matches the OS ID provided. Within a matching section, it searches for the provided key and skips (thus effectively removing it) writing it to a temporary file that will replace the original.

2.0.902 Technical Description

```
name: os_config_undefine_key

description: Deletes a specific key from a specific OS configuration in a configuratio

globals: [ os_conf: The path of the operating system configuration file ]

arguments: [ $1: OS ID to search for in the configuration file, $2: Key to remove from t

outputs: Rewrites the OS configuration file without the specified key for the identifi

returns: 0 if the function completes successfully. 1 if the configuration file does not

example usage:
$ os_config_undefine_key "ubuntu" "serverName"
```

2.0.903 Quality and Security Recommendations

1. Consider implementing error checking to validate the input data types.
2. Double-check file permissions and ensure the script cannot be executed with root privileges unnecessarily to mitigate security risks.
3. Implement more robust error handling; do not just return if the OS configuration file does not exist.
4. Consider backup and recovery strategies for your configuration files in case of accidental removal of crucial configuration entries.
5. Validate the given OS ID and key against a predefined whitelist to prevent potential misconfiguration or injection attacks.

2.0.904 `os_config_undefine_section`

Contained in `lib/functions.d/os-functions.sh`

Function signature: c707b545b7e0afa23ad74c42f17409f2e6d5963aafd2e0d2f88098f2a465b5cb

2.0.905 Function overview

The `os_config_undefine_section` function is used to remove a specific section from the Operating System's configuration in a safe manner. It identifies the section by a provided `os_id` passed as an argument. It then reads through the configuration file line by line, and if the line corresponds to the header of the identified section, it flags processing to skip recording the lines that correspond to that section into a temporary file. After processing the entire configuration file in this manner, it replaces the original configuration file with the temporary file, effectively removing the specified section from the configuration file without disrupting the remaining content.

2.0.906 Technical description

- **name:** os_config_undefine_section
- **description:** This function removes a specified section from the operating system's configuration file.
- **globals:** [`os_conf`: the operating system's configuration file, `in_section`: a flag to indicate whether a line belongs to the target section or not, `temp_file`: a temporary file to store the configuration excluding the target section]
- **arguments:** [`$1: os_id`: the identifier for the section to be removed in the OS configuration]
- **outputs:** The function does not print any output, it modifies the OS configuration file.
- **returns:** If the OS configuration file does not exist it returns `1`, else after successfully updating the configuration file it returns `0`.
- **example usage:** `os_config_undefine_section CentOS`

2.0.907 Quality and security recommendations

1. Consider validating the input `os_id` before proceeding with the function. It should not be empty or contain dangerous characters.
2. Naming your variables more descriptively (`in_section` to `in_target_section`) can help improve code legibility.
3. Ensure that the temporary file deletion is handled appropriately to avoid potential security vulnerabilities.
4. Review your error handling - for example, consider if `0` is an appropriate return value if the `os_id` provided does not exist in the file.
5. Be sure to set proper permissions on the original and temporary configuration files to prevent unauthorized access or edits.

2.0.908 `os_config_validate`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: cff9e5d9494106a1ac4e9f0dec1bccd9a2b35e924bf70272149ee31c6ae35e38

2.0.909 Function overview

The `os_config_validate` function is a bash function that is used to validate whether the required fields such as 'hps_types', 'arch', 'name', 'version' and 'status' for an operating system specified by the `os_id` argument exist. If they do not exist, the function will append them to the `missing_fields` array and set `valid` to 1. The function echoes an error message if the `os_id` does not exist or if any required fields are missing and returns the value of `valid`.

2.0.910 Technical description

Name: `os_config_validate`

Description: The function validates whether the required fields of an operating system specified by the `os_id` argument exist.

Globals: None

Arguments:
- `$1: os_id` - An identifier for the operating system. - `required_fields: Array` - List of required field names. It's a local variable and defined inside the function.

Outputs: Error message if the OS does not exist or if there are missing required fields.

Returns: `$valid` - 0 if all required fields exist, 1 if there are missing fields.

Example usage: `os_config_validate ubuntu`

2.0.911 Quality and security recommendations

1. Implement error checking for the field names in the required_fields array to ensure that they are valid.
2. Safeguard system commands such as 'echo' by using their full path.
3. Consider adding input validation to the os_id and other potential arguments for enhanced security.
4. Add comments in code as much as possible for future readability and maintainability.
5. Ensure that the array `required_fields` does not contain sensitive data (like passwords), if it does, then it's advisable to handle such information in a secure manner.

2.0.912 `os_get_latest`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: a3e4b8498edbef98bbab321be3cedf7a513241b2922526887a3b4169195c711e

2.0.913 Function Overview

The function `os_get_latest()` is designed to interrogate a configuration file and retrieve the most recent version of a particular operating system. It takes in the operating system's name as a parameter and searches through a configuration file for matches. When a match is found, the function compares versions to identify the most recent one. If a most recent version is found, the function echoes the operating system's id configuration (comprising the architecture, name, and version) and returns with a success code. If no recent version is identified, or if the configuration file does not exist, it returns with a failure code.

2.0.914 Technical Description

- Name: `os_get_latest()`
- Description: This function retrieves the latest version of an operating system given its name.
- Globals: None.
- Arguments:
  - `$1`: The name of the operating system to search for.
- Outputs:
  - If the latest version of the OS is found, the function echoes the OS's id in the format of architecture:name:version.
  - If no latest version is found or the configuration file does not exist, no output is produced.
- Returns:
  - `0`: If a latest version corresponding to the OS name parameter was found.
  - `1`: In all other cases, such as when the configuration file is missing or no version of the OS name in the parameters is identified.
- Example usage:

```
os_get_latest "ubuntu"
```

2.0.915 Quality and Security Recommendations

1. Maintain up-to-date versions of Bash to ensure all constructions used in this function work as expected.
2. Treat the input OS name cautiously. It's always a good practice to sanitize user input to prevent possible attack vectors, like Bash injection attacks.
3. Use a consistent file-path naming convention for the OS configuration file (_get_os_conf_path). Consistency measurably reduces complexity and increases maintainability and the speed of development.
4. Make sure the OS configuration file is kept secure. It should have restricted access as it contains information about different versions of operating systems.
5. Implement error handling in case of any failure in reading the configuration file or if the file doesn't exist.
6. Consider using a logging mechanism to keep track of the function's activity for tracing and debugging purposes.

2.0.916 `os_id_to_distro`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: d8ff171b8256ab5df7e39e84cfaa1576a9c878859fe9b5b0561fe29db548e50f

2.0.917 Function Overview

The `os_id_to_distro` function converts an OS identifier into a distribution string format. It takes OS details in the format `arch:name:version` and transforms it into a string format `arch-linux-osname-version`. It accepts an OS identifier string as an argument, parses it into respective OS parameters (architecture, name, and version), and then assembles them into the desired format. If no argument is given, it logs an error and returns.

2.0.918 Technical Description

- Name: `os_id_to_distro`
- Description: This function converts an OS identifier into a distribution string format. It receives the parameters arch, name, and Version in a unique string separated by ':' and outputs them in a new format after adding `-linux-` in between.
- Globals: No global variables used.
- Arguments:
  - `$1: os_id` - A string containing the OS ID in the format `arch:name:version`.
- Outputs: Prints out a string in the format `arch-linux-osname-version`.
- Returns: Returns 1 if the argument (os_id) is not provided.
- Example usage:

```
os_id_to_distro "x86_64:alpine:3.20"    # Output:
 ↪  x86_64-linux-alpine-3.20
```

2.0.919 Quality and Security Recommendations

1. Include comprehensive error handling to accustom for cases when the input string does not follow the expected `arch:name:version` format.
2. Add a check to validate that the architecture, osname, and version values are valid and standardized before operating on them.
3. Implement unit tests to ensure function behavior remains correct after modifications.
4. Document the function using Comment Standards to maintain clarity and to make sure subsequent users understand its implementation and usage.
5. If the function will be used in larger scripts or in a sensitive environment, consider implementing logging for debugging and reversal information.

2.0.920 `_osvc_cluster_agent_key`

Contained in `lib/functions.d/opensvc-function-helpers.sh.sh`

Function signature: 8ad0d0f636d57d1079f22f7672816963096f75042862926e8849b3f57d049752

2.0.921 Function overview

The `_osvc_cluster_agent_key` function is a Bash function developed to enforce a key policy for the OpenSVC cluster agent. This function reads the cluster key and the existing on-disk key and handles mismatches or missing keys accordingly. If a mismatch is detected between the disk key and the cluster key, an error is logged and the function returns. If there is no cluster key, the function adopts the existing disk key or generates a new key.

2.0.922 Technical description

- **name**: _osvc_cluster_agent_key
- **description**: This function reads the cluster key and the existing on-disk key. If a mismatch is detected, an error is logged and the function returns. If there is no cluster key, the function adopts the existing disk key or generates a new one.
- **globals**: [`key_file`: Holds the path for the agent key]
- **arguments**: [`$1`: Not used in this function, `$2`: Not used in this function]
- **outputs**: Various log messages, writes the cluster key to a key file, and modifies the cluster configuration.
- **returns**: 2, if there is a mismatch between the cluster key and the disk key; 0, in all other cases.
- **example usage**: `_osvc_cluster_agent_key`

2.0.923 Quality and security recommendations

1. To improve code readability, avoid unnecessary comments. Code can be self-explanatory with the appropriate variable and function names.
2. Replace the use of `2>/dev/null`; instead use proper error handling.
3. Make use of readonly variables, where applicable, to enhance security.
4. The function might fail if the permissions of the `key_file` are not correctly set. Thus, add some checks for the file permission.
5. Consider adding more error control and checking of return values to detect possible execution failures.
6. For added security, validate the format of the `cluster_key` before overwriting it to the `key_file` to prevent potential exploits or misuses.

2.0.924 `_osvc_cluster_secrets`

Contained in `lib/functions.d/opensvc-function-helpers.sh.sh`

Function signature: b28fddffee012858f4aa8ec1b9e77088bd56dcabeecfdfcd2c823c38c282466f

2.0.925 Function Overview

The function `_osvc_cluster_secrets()` is used to generate a secret for a cluster. It first attempts to retrieve a cluster secret. If the secret does not exist, it then tries to generate one using the `openssl` command. If the `openssl` command is not available, it generates a secret using a combination of the `tr`, `head`, and `urandom` commands. If the generation fails, it logs an error and returns with a non-zero exit status. Otherwise, it sets the newly generated secret to the cluster configuration, logs an informational statement about it, prints the secret, and exits with status 0.

2.0.926 Technical Description

- **Name:** `_osvc_cluster_secrets`
- **Description:** This function generates or retrieves a secret for an OpenSVC cluster. If the `openssl` command is available, it is used to generate the secret; otherwise, it resorts to shell built-in commands.
- **Globals:** No global variables are used.
- **Arguments:** No parameters are passed to this function.
- **Outputs:** The function outputs the cluster secret.
- **Returns:** Returns 1 if it fails to generate a cluster secret. Otherwise, returns 0.
- **Example usage:**

```
cluster_secret=$(_osvc_cluster_secrets)
```

2.0.927 Quality and Security Recommendations

1. Error handling could be improved. Currently, the function fails silently if it cannot fetch the cluster secret, and it could be difficult to trace what happens when cluster_config yields an error.
2. Consider increasing the randomness of the `tr` method to generate a secret. Right now, it only uses a-f and 0-9, which may be easily brute-forced.
3. Validate that the `openssl` command is available on the system before declaring the secret. If not, alternate secure methods of generating random numbers should be implemented.
4. Usage of random number generators could be standardized to one method rather than having two methods depending on the availability of `openssl`.
5. It would be beneficial to return unique exit statuses for different errors to help with debugging.

2.0.928 `_osvc_config_update`

Contained in `lib/functions.d/opensvc-function-helpers.sh.sh`

Function signature: cf1d9f1aba241a37cf2273a4b5fb25f26ee20ad05d70c8c971420b921a9e2118

2.0.929 Function overview

The function `_osvc_config_update()` is used to update a cluster configuration in the OpenSVC system. It accepts key=value pair arguments which describe the settings to be updated in the OpenSVC cluster configuration.

2.0.930 Technical description

- **name**: `_osvc_config_update`
- **description**: This function updates a cluster configuration in the OpenSVC system. The function first checks if at least one key=value pair has been provided as input. If not, the function logs an error and returns 1. If such pairs are provided, the function updates the OpenSVC cluster configuration accordingly and logs this activity. Ideally, the function should return 0 if the update is successful. However, it returns 1 if an error occurs while updating.
- **globals**: [`set_args`: Local array that stores key=value pairs for configuration update]
- **arguments**: [`$@`: Accepts one or more key=value pairs that specify the settings to be updated]
- **outputs**: Logs activity and error messages
- **returns**: 0 if the configuration update is successful and 1 if it fails or if no key=value pairs have been provided as input
- **example usage**: `_osvc_config_update "key1=value1" "key2=value2"`

2.0.931 Quality and security recommendations

1. Implement data validation for the key=value pairs received as input.
2. Separate logging and return statements for better modularity.
3. Implement a mechanism to restore the old configuration if the function fails to update the new configuration.
4. Consider using more descriptive and specific debug and error messages.
5. Implement input sanitization to prevent script injection or other malicious activities.

2.0.932 `osvc_configure_cluster`

Contained in `lib/functions.d/opensvc-functions.sh`

Function signature: 879172429a557503678b7818e10ab4bd855bd4ffddb139783fed8879d62d7e1d

2.0.933 Function overview

The function `osvc_configure_cluster()` is designed to configure an OpenSVC cluster. This involves setting up the IPS node identity, retrieving cluster configura-

tion, setting up heartbeat configuration, constructing configuration update arguments, handling cluster secrets, applying all configuration updates, creating heartbeat secrets, verifying the application of the updated configuration, and logging the successful configuration of the cluster.

2.0.934 Technical description

- **name**: `osvc_configure_cluster`
- **description**: This function configures settings related to an OpenSVC cluster. It sets up the IPS node identity, retrieves cluster configuration, handles cluster secrets and applies several updates to cluster configuration.
- **globals**: [`osvc_nodename`: the name of the OpenSVC node, `cluster_name`: the name of the cluster, `cluster_secret`: secret code for cluster, `hb_type`: the type of heartbeat process for the cluster]
- **arguments**: None.
- **outputs**: This function mainly outputs logs about its progress, with information about the cluster's configuration, node, heartbeat type, config filename, and agent key policy status.
- **returns**: It returns 0 for successful execution. If any step fails, it would return 1.
- **example usage**: `bash osvc_configure_cluster`

2.0.935 Quality and security recommendations

1. Add input validation. The function currently does not validate the retrieved cluster configuration and heartbeat configuration values. It is recommended to add validation steps to ensure that these values are valid and safe to use.
2. Uncomment and correct the cluster agent key policy section of code for enhanced security.
3. Consider error handling when retrieving cluster configuration and heartbeat configuration. Adding appropriate error handling can help prevent unexpected behavior and also make debugging easier.
4. Uncomment and correct the functions for creating heartbeat secrets and setting up the cluster secret for better security.
5. Implement better handling when communication with the OpenSVC daemon fails for improved reliability.
6. Improve log messages by including actionable steps and more specific error information, aiding in better troubleshooting.
7. Consider breaking up the function into smaller functions. This would achieve clearer, more modular code that is easier to maintain and test.

2.0.936 `_osvc_create_conf`

Contained in `lib/functions.d/opensvc-function-helpers.sh.sh`

Function signature: fbe36f7691006261555525550c96682a9254f547269c1a2322ffb8e549c857b7

2.0.937 Function overview

The `_osvc_create_conf()` function in Bash is used to automatically generate an opensvc.conf configuration file in the /etc/opensvc directory. It does this by first initializing the desired output file path and logging its current activity. Then it fetches the DNS domain configuration from cluster_config, creating an error log and terminating the function if the domain setting cannot be fetched. Next, it creates a temporary tempfile with a unique name, logging an error and terminating if this step fails. It then writes a minimal configuration to the temp file and atomically moves the temp file to the set path for the configuration file. The permissions of the new configuration file is set to readable and writable by the owner and readable by the group and others, and the ownership is set to root:root. Afterwards, the function logs its successful generation of the config file and ends.

2.0.938 Technical description

- **name**: `_osvc_create_conf`
- **description**: This function generates a opensvc.conf configuration file at the path /etc/opensvc/.
- **globals**: [dns_domain: The domain setting fetched from cluster_config]
- **arguments**: [None]
- **outputs**: Logs messages regarding its processes and errors.
- **returns**: Failure (1) if a DNS_DOMAIN is not set in cluster_config or if it fails to create a temp file. Otherwise, success (0).
- **example usage**: `_osvc_create_conf`

2.0.939 Quality and security recommendations

1. Consider updating function to handle errors and edge cases more explicitly, such as when the /etc/opensvc directory doesn't exist or is not writable.
2. Ensure that the DNS_DOMAIN setting in cluster_config cannot be manipulated by unauthorized users to prevent potential hacks.
3. Consider setting stricter permissions on the generated opensvc.conf to minimize potential breaches.
4. Check if cluster_config is present and accessible before calling this function.
5. Consider character escaping or validation to avoid potential bugs or security issues from unexpected DNS_DOMAIN inputs.

2.0.940 `_osvc_create_hb_secrets`

Contained in `lib/functions.d/opensvc-function-helpers.sh.sh`

Function signature: aed67edfeb1fd1a63d3e83b7088b23dd0bd8242d25f17ebe70d11e474348804d

2.0.941 Function overview

The bash function `_osvc_create_hb_secrets` checks if a heartbeat secret object exists, checks for the presence of a secret key, creates this secret object if it does not exist, generates a random secret for heartbeat authentication and adds it to the object. The function also verifies the created heartbeat secret.

2.0.942 Technical description

- **Name**: `_osvc_create_hb_secrets`
- **Description**: This function checks for the presence of a heartbeat secret object, creates one if it does not exist, generates a random secret, and adds it to the object. After addition, it verifies the function of the secret.
- **Globals**: None
- **Arguments**: No directly passed arguments
- **Outputs**: Logs various debug, info and error messages about the status and process of heartbeat secret object creation and verification.
- **Returns**: 0 if the heartbeat secrets are successfully configured, 1 if any error occurs during the process.
- **Example usage**:

```
_osvc_create_hb_secrets
```

2.0.943 Quality and security recommendations

1. This function does not use any command line arguments. It does not have any form of argument sanitization or validation as there are no arguments.
2. This function relies heavily on the commands `om` and `openssl`, check that these system utilities are available and reliable in the bash environment.
3. For system hardening, the error redirections `>/dev/null 2>&1` and `2>/dev/null` could potentially hide valuable debugging information. Consider more granular log verbosity controls.
4. For security considerations, investigate how secrets are transmitted, to avoid leakage.
5. Review the error handling mechanisms to ensure failed function calls are properly handled.
6. The random secret key generation using /dev/urandom could be examined to ensure that it meets security best practices.
7. Ensure that all 'hps_log' output is correctly routed and secured, as it might contain critical system information.

2.0.944 `osvc_create_services`

Contained in `lib/functions.d/opensvc-functions.sh`

Function signature: cd2be3b7734b0eaad20f9892d53890d9f51dca5c8b5d82c54086a4e9b1fa2381

2.0.945 Function overview

The `osvc_create_services` function is responsible for creating and configuring an OpenSVC cluster. It is a shell function that updates configurations, verifies the success of updates, logs error if the updates fail and initializes, edits, sets parameters, provisions, and starts an `iscsi-manager` service.

2.0.946 Technical description

- **Name**: `osvc_create_services`
- **Description**: This function is used to create and configure an OpenSVC cluster with the service `iscsi-manager`. It applies configurations and logs an error if the configuration fails. Thereafter, it initializes, edits, sets parameters, provisions, and starts the `iscsi-manager` service.
- **Globals**: None
- **Arguments**: None
- **Outputs**: Logs an error if there's a failure in configuring the OpenSVC cluster.
- **Returns**: 1 if the service fails to configure the OpenSVC cluster.
- **Example usage**:

  ```
  osvc_create_services
  ```

2.0.947 Quality and security recommendations

1. To prevent errors, ensure that all scripts and commands in the function have been properly validated before this function is called.
2. Check error handling specifically, when configuration updates fail. Consider implementing retry logic or cleanup actions.
3. For security, consider implementing permissions checks on commands that are executed, ensuring that they are run with the minimum necessary permission level.
4. Ensure proper logging is done at every step of function execution to enable easier debugging.
5. Encapsulate the function logic inside error handlers to ensure graceful termination of the script in the event of an uncaught error.
6. Consider making the function receive its dependencies as arguments, this will make testing easier and the code will be more solid.

2.0.948 osvc_create_services

Contained in lib/functions.d/opensvc-functions.sh

Function signature: cd2be3b7734b0eaad20f9892d53890d9f51dca5c8b5d82c54086a4e9b1fa2381

2.0.949 Function Overview

The function osvc_create_services is responsible for creating and configuring an OpenSVC cluster with an iSCSI manager. It starts by defining a local array to hold configuration updates, adding the needed cluster name to the array. It then attempts to apply all configuration updates via the _osvc_config_update function. After successfully configuring the OpenSVC cluster, the function goes on to create, edit, set parameters and start the iSCSI manager using the om command.

2.0.950 Technical Description

- **name**: osvc_create_services
- **description**: The function creates and configures an OpenSVC cluster with an iSCSI manager.
- **globals**: None
- **arguments**: None
- **outputs**: An error message is output if there is a failure configuring the OpenSVC cluster.
- **returns**: 1 in case of a failure to configure the openSVC cluster. Otherwise, it does not return anything.
- **example usage**: osvc_create_services

2.0.951 Quality and Security Recommendations

1. Incorporating error handling or sanity checks would ensure that all om command invocations are successful and report back correctly in case they fail.
2. It's useful to validate that the cluster name is properly set and isn't blank or containing illegal characters.
3. Ensure sensitive information such as possible cluster credentials or keys are kept secure and not openly displayed or annotated within the script. For instance, consider the need of using system variables or secure external files to hold such information if needed.
4. Assess whether the script should continue its execution or exit if certain components cannot be created or started. This will avoid running an incomplete setup which could lead to issues later.
5. Depending on the environment the script is written for, additional logging may be beneficial for auditing and troubleshooting purposes.

2.0.952 _osvc_get_auth_token

Contained in lib/functions.d/opensvc-function-helpers.sh.sh

Function signature: 1eaf937579353ed95dcc7ba9300c47e601a9f35bddea106967009534acde2413

2.0.953 Function overview

The _osvc_get_auth_token() function is a bash function designed to retrieve an authentication token. This token is obtained by the OpenSVC daemon for the specific role join, within a limited duration of 15 seconds. Currently, there is a placeholder in the code for adding the hostname of the calling subject.

2.0.954 Technical description

- **Name**: _osvc_get_auth_token
- **Description**: This function retrieves an authentication token with a role of join and a duration of 15 seconds from the OpenSVC daemon.
- **Globals**: None
- **Arguments**: None
- **Outputs**: Prints the authentication token generated by the OpenSVC daemon.
- **Returns**: Returns 0 on successful execution, otherwise it will return an error code given by the om command.
- **Example usage**: _osvc_get_auth_token

2.0.955 Quality and security recommendations

1. Input validation: Even though this function doesn't currently take any arguments, checks should be built into the function in case arguments are added in the future.
2. Error handling: Better error handling could be implemented. For example, if the om command cannot execute, the function might fail with a non-descriptive error.
3. Logging: Consider adding logs in order to make debugging easier in case of problems.
4. Security: Consider restricting the permissions of the script containing this function to prevent unintended access. Also, careful handling of the authentication token is essential to prevent any security vulnerabilities.

2.0.956 _osvc_get_auth_token

Contained in lib/functions.d/opensvc-function-helpers.sh.sh

Function signature: 1eaf937579353ed95dcc7ba9300c47e601a9f35bddea106967009534acde2413

2.0.957 Function overview

The function `_osvc_get_auth_token` is intended to generate an authentication token by interacting with a system daemon. The function does not accept any parameters and does not specifically declare any global variables. Currently, the token duration is hardcoded to 15 seconds, and the token role is set to 'join'. There may be further enhancements to be considered.

2.0.958 Technical description

Name: `_osvc_get_auth_token`

Description: This function echoes an authentication token. It directly interacts with a daemon controller `om` to generate this. The token has a hard-coded duration of 15 seconds and is assigned the role 'join'.

Globals: None.

Arguments: None.

Outputs: Sends the authentication token to STDOUT.

Returns: The output of the `om daemon auth` command, or an error message if the command fails.

Example usage:

```
token=$(_osvc_get_auth_token)
```

2.0.959 Quality and security recommendations

1. **Use Conditional Checks**: Add error handling to check if the `om` command succeeds, and whether the daemon is running or not before executing the function.
2. **Parameterize Function**: Instead of hardcoding the values '15s' and 'join', consider allowing them as parameters to make the function more versatile.
3. **Avoid Echoing Token to STDOUT**: Echoing sensitive information such as authentication tokens to STDOUT can be a security risk. Consider an alternative method to handle the token.
4. **Documentation**: The function could use inline comments to clarify what each step in the function does. Commenting the code would help maintainers and developers understand the function better.
5. **Use of Global Variables**: Currently the function does not declare any global variables. Be cautious if the function will access global variables in future, as this could potentially introduce side effects making the function less predictable.
6. **Hosting Machine**: Running daemon processes in a separate machine may be a safer design strategy, since co-hosting them with other key processes can introduce resource contention and security risks.

2.0.960 `_osvc_kv_set`

Contained in `lib/functions.d/opensvc-functions.sh`

Function signature: e83bb88e314eb7098eb2ed8868cdecf150f3a7c4cec631f0ac14eadabbeaa0d3

2.0.961 Function overview

The function `_osvc_kv_set()` is a local key-value setter. This bash function sets a key-value combination in the OSVC configuration file. It uses the local variable k to represent key, and v to represent value. These are then passed to `om config set` with the `--kw` option.

2.0.962 Technical description

- **Name**: `_osvc_kv_set`
- **Description**: This function takes a key and a value as inputs, and sets this pair in the OSVC configuration, by using `om config set` command.
- **Globals**: None.
- **Arguments**:
  - `$1: key` - This argument represents the key that we need to set.
  - `$2: value` - This argument represents the corresponding value for the key.
- **Outputs**: Executes `om config set` with `--kw` option and the passed key-value pair.
- **Returns**: None. The changes are made in the configuration.
- **Example usage**: `_osvc_kv_set "myKey" "myVal"`

2.0.963 Quality and security recommendations

1. Check for null or empty key-value pairs: Before setting a key-value pair in the configuration, it should be validated that neither the key nor the value is null or empty. This could be done as a guard clause within the function.
2. Avoid overriding of existing keys: While setting a new key-value pair, ensure that the key does not already exist to prevent unintentional overwriting.
3. Input sanitization: Ensure that key and value inputs do not contain characters that could potentially exploit script vulnerabilities or disrupt normal operation.
4. Configuration file permissions: Check for necessary permissions before updating the configuration file. If the permissions are not sufficient, the function should either notify the user or fail gracefully.

2.0.964 `osvc_prepare_cluster`

Contained in `lib/functions.d/opensvc-functions.sh`

Function signature: b3056c9c6999f4845c0dbfe6d11d0957e9a75e1285f61a792b08d5b8e9a56380

2.0.965 Function Overview

The `osvc_prepare_cluster` function is part of a Bash script that sets up an OpenSVC cluster for interacting with the IPS system. The script does the following:

1. Checks if OpenSVC is installed on the system. If not, it logs an error message and stops executing.
2. Sets up the OpenSVC directory structure, and if it fails, an error message is logged and the function stops executing.
3. Creates `opensvc.conf` configuration file. If creation fails, an error message is logged and the function stops executing.

2.0.966 Technical Description

- **Name:** osvc_prepare_cluster
- **Description:** This function prepares an OpenSVC cluster for interaction with the IPS system by ensuring the OpenSVC is installed, sets up directory structures, and creates a configuration file.
- **Globals:** None
- **Arguments:** None
- **Outputs:** Logs informational, success, and error messages during its execution.
- **Returns:** Returns 1 if any operation fails, otherwise no explicit return.
- **Example usage:** `osvc_prepare_cluster`

2.0.967 Quality and Security Recommendations

1. Consider adding validation checks for the existence of functions: `ensure_opensvc_installed`, `_osvc_setup_directories`, and `_osvc_create_conf` before their usage.
2. Implement exception handling and cleaning mechanism if the script terminates unexpectedly.
3. Document outputs and potential error messages more clearly to improve readability and maintainability.
4. Make sure that all logs, especially error logs, provide actionable information.
5. The logged error messages should not expose sensitive data such as file paths, IPs, etc.

2.0.968 `osvc_process_commands`

Contained in `lib/functions.d/opensvc-function-helpers.sh.sh`

Function signature: ba74ea35c4b9ae28ad246acbd28f8ccba4242f86e167d3e986d76f51355d67f3

2.0.969 Function Overview

This bash function `osvc_process_commands()` is primarily designed to process a specified command. It takes as an input (`$1`) a command and checks if it's valid or not. If no command is passed or an unknown command is provided, it logs an error and returns `1`. If the command is `get_auth_token`, it delegates to a private function `_osvc_get_auth_token`.

2.0.970 Technical Description

- **Name:** osvc_process_commands()
- **Description:** The function processes and validates provided commands. It uses the local `cmd` variable to store the command argument and uses `case` conditional constructs to check whether the command is `get_auth_token` or not. If `get_auth_token`, it calls `_osvc_get_auth_token` function; otherwise, it logs an error message and returns `1`.
- **Globals:** None
- **Arguments:** `$1`(command): The command to be processed by the function.
- **Outputs:** Two potential error messages can be logged: "No command specified" and "Unknown command"
- **Returns:** It either returns `1` on failure (if the command is not provided or unknown command is provided) or moves to the relevant function without explicitly returning a value.
- **Example Usage:**

```
osvc_process_commands "get_auth_token"
```

2.0.971 Quality and Security Recommendations

1. Validate the input more thoroughly: While the function does check if a command parameter is provided, it doesn't validate the parameter further.
2. A more descriptive error message should be displayed for unknown commands. The name of the unknown command could be included for easier troubleshooting.
3. It would be better to define exit status codes at the top of the script or the function. This can improve the readability of our script by giving meaningful names to our exit statuses.
4. Ensure logging is properly set up and error messages are logged at appropriate levels.
5. Need to be careful with command injection attacks. Be careful not to accept any command without proper validation.
6. Consider limiting the scope of the script to handle only expected commands to mitigate the risk of inadvertent script execution.

2.0.972 _osvc_setup_directories

Contained in lib/functions.d/opensvc-function-helpers.sh.sh

Function signature: 7a031e0402487c7f287dfc1d0a03b9c851605cdb0946a9dddb2377b5798a0b42

2.0.973 Function overview

This bash function, _osvc_setup_directories, is designed to set up essential directories for the OpenSVC system. It works by creating directories at given paths (/etc/opensvc, /var/log/opensvc, and /var/lib/opensvc). If these directories already exist, nothing happens. However, if any of these directories do not exist, they are created. If there are any failures in the directory creation process, an error is logged and the function returns 1. If the directories are created successfully, the function logs this and returns 0.

2.0.974 Technical description

- **Name**: _osvc_setup_directories
- **Description**: This function is used to create necessary OpenSVC directories at specific paths. If directory creation fails, it logs an error and returns 1. If it succeeds, it logs success and returns 0.
- **Globals**: None
- **Arguments**: None
- **Outputs**:
  - Logs an error message if directory creation fails
  - Logs a debug information message if directory creation succeeds
- **Returns**:
  - Returns 1 if directory creation fails
  - Returns 0 if directory creation succeeds
- **Example usage**: _osvc_setup_directories

2.0.975 Quality and security recommendations

1. Before creating directories, one might consider checking if there is sufficient disk space on the system to avoid improper directory creation.
2. Be sure that the directories are being created with the correct permissions to prevent potential security vulnerabilities.
3. Consider checking if the directories you're attempting to create already exist before creating them again.
4. Use more definitive logging messages which includes the directory name in case of any failure / success as this aids system administrators in easier debugging.
5. Always consider error handling. If a necessary directory is unable to be created, the application shouldn't proceed as if it were created. This check can help avoid insidious bugs and tighten system integrity.

2.0.976 `_osvc_verify_daemon_responsive`

Contained in `lib/functions.d/opensvc-function-helpers.sh.sh`

Function signature: 1e92c3b03dd26b02df08a43e8209d3e397e16970cf0ade303db8baf1f7c06edb

2.0.977 Function Overview

The function _osvc_verify_daemon_responsive is designed to validate the responsiveness status of the OpenSVC daemon in a Bash environment. It begins by logging a debug message indicating the start of the verification process. Next, it checks the status of the cluster managed by the OpenSVC daemon. If the daemon is responsive and the cluster status check is successful, it logs a debug message indicating the same and returns a 0. Otherwise, it logs an error message and exits the program with a non-zero exit status.

2.0.978 Technical Description

- **Name:** _osvc_verify_daemon_responsive
- **Description:** This function checks the responsiveness of the OpenSVC daemon.
- **Globals:** None
- **Arguments:** None
- **Outputs:** Debug or error logs related to OpenSVC daemon responsiveness.
- **Returns:** 0, if the OpenSVC daemon is responsive; otherwise, it terminates the program by executing exit 1.
- **Example usage:**

```
_osvc_verify_daemon_responsive
```

2.0.979 Quality and Security Recommendations

1. Add more detailed logging: To assist debugging and maintenance, consider adding more detailed logging messages. For example, any data or metrics regarding the daemon's responsiveness might be useful.
2. Check command success: Always check if an external command was successful or not before proceeding to the next statement. In this script, there are statements like 'om cluster status'; make sure to add error handling for them.
3. Exit status: Exit status should be different for all different types of errors, here status 1 has been used to indicate unresponsiveness of OpenSVC daemon. Different exit status for different errors can be more helpful in troubleshooting.
4. Function Documentation: Add comments in the function to explain what each part of the function is doing.
5. Input Validation: The function does not take any input from others. So, there are no validation concerns in this function. But if changes are made and inputs are required, validate all inputs to maintain the integrity of the function.

2.0.980 _osvc_wait_for_sock

Contained in lib/functions.d/opensvc-function-helpers.sh.sh

Function signature: c9fc88259b4aeaee64ce5f076d41e886e9192aba5fa2041b4ece31bdf991f1d7

2.0.981 Function Overview

This function, _osvc_wait_for_sock(), is specifically designed to wait for the OpenSVC daemon socket. It uses a loop to check up to 10 times if the OpenSVC daemon socket is ready. The loop waits for 1 second between each check. If the daemon socket is ready, it logs the status and returns 0, signifying successful execution. If the daemon socket is not ready after 10 seconds, it logs an error and returns 1, indicating that the function did not complete successfully.

2.0.982 Technical Description

- **Name:** _osvc_wait_for_sock
- **Description:** This function waits for the OpenSVC daemon socket to be ready, checking 10 times with a 1 second gap.
- **Globals:** None used.
- **Arguments:** This function does not have any arguments.
- **Outputs:** Logs either a debug message when the daemon socket is ready or an error if it is not ready after 10 seconds.
- **Returns:** Returns 0 if the daemon socket is ready. Otherwise, it returns 1.
- **Example usage:**

```
_osvc_wait_for_sock
```

2.0.983 Quality and Security Recommendations

1. Improve error handling: Consider increasing the number of retry attempts to accommodate network latency or system load times that could delay the daemon socket readiness.
2. Enhance logging: Provide more detailed messages for the logging, like logging the current iteration of the loop.
3. Parameterize hardcoded values: The number of attempts (10) and sleep time (1 second) are hardcoded. Consider moving these to configurable variables.
4. Sock file security: Ensure proper permissions are set on the socket file /var/lib/opensvc/lsnr/http.sock to prevent unauthorized access.
5. Check for the presence of “hps_log” function: Before calling hps_log, it would be safer to check if this function exists and is executable. It can prevent potential execution halt if the function is missing for any reason.

2.0.984 o_task_create

Contained in lib/functions.d/o_opensvc-task-functions.sh

Function signature: df1ceaa02ef47a840c972444f435e795d063a2e6dd654e1a3103e7ddb37e57ab

2.0.985 Function Overview

The o_task_create function creates or updates existing service tasks. It accepts four parameters, namely service_name, task_id, command, and nodes. The function first validates these parameters, then checks the command for any unsupported quote characters, and finally checks if the service already exists. If the service exists, the function will update the task. Otherwise, it will create a new service with the task in a single atomic operation. After these steps, the error states for the service are cleared and the service is unfrozen.

2.0.986 Technical Description

- Name: o_task_create
- Description: This function is designed to create or update tasks in a service. The function validates inputs, checks for unsupported quote characters and checks whether the service already exists. It updates the service if it exists or creates a new service with task otherwise. After these operations, it unfreezes the service and clears the error states.
- Globals: None
- Arguments:
  - $1: service_name - The name of the service
  - $2: task_id - ID of the task to create or update
  - $3: command - The intended command to be issued
  - $4: nodes - The nodes where the command should run
- Outputs: Logging to stdout/stderr regarding process status. The function may communicate issues with the input parameters or state whether the service exists or not. It will also provide status on freezing, clearing error states and task creation.
- Returns: 0 if the function succeeds, otherwise it returns 1.
- Example Usage:

```
$ o_task_create "service01" "task01" "echo hello" "node1 node2
↪  node3"
```

2.0.987 Quality and Security Recommendations

1. Add more specific validation tests for the inputs. For instance, the cmd might be better validated with a regex to prevent security issues.

2. Implement error handling for each command executed within the function to ensure robustness.
3. The function should not silently suppress errors by redirecting stderr to null, this diminishes the opportunity for troubleshooting in case issues arise.
4. Check if the `nodes` parameter contains unsupported characters to ensure system stability and integrity.
5. Add capability to handle escape characters in the command parameter.
6. Conduct testing based on a security-first principle, especially when updating or adding new features to the function. Use fuzzing or other comprehensive security testing methods.

2.0.988 `o_task_delete`

Contained in `lib/functions.d/o_opensvc-task-functions.sh`

Function signature: a414368fa84dd9a7cc8a27adeeff9ad30e2ff155d1a179a3c949ab7bc74873bc

2.0.989 Function Overview

The function `o_task_delete` performs the operation of deleting a specified task from a service. The function takes two parameters, `service_name` and `task_id`. The `service_name` is mandatory and if it's not provided or nonexistent, the function logs an error message and return 1. If the `task_id` is not provided, the function deletes the entire service. If it is provided, the function tries to delete that specific task from the service.

2.0.990 Technical Description

- **Name:** `o_task_delete`
- **Description:** This function is responsible for deleting specified tasks or an entire service if task is not specified, from a provided service. It validates the service name, checks if the service exists, and either deletes the entire service or a specific task from that service.
- **Globals:** None
- **Arguments:**
  - `$1: service_name`, Name of the service from which task is to be deleted.
  - `$2: task_id`, ID of the specific task to be deleted.
- **Outputs:** Log messages indicating the deletion operations/conditions.
- **Returns:** The function returns 0 on successful execution, and 1 when either service doesn't exist or fails to delete the service or task.
- **Example usage:** `o_task_delete "my_service" "task1"`

2.0.991 Quality and Security Recommendations

1. Input validation: Although the function validates `service_name`, it does not validate `task_id`. This leaves room for unintentional behavior if non-existing task ID is provided.
2. Error handling: The function does a good job of logging the errors and warning messages, but for better usability, it could also throw exceptions or handle errors in a more user-friendly manner.
3. Eliminate duplicate code: There are multiple instances where the function checks the existence of a service. These could possibly be combined and modularized to improve clarity and avoid code repetition.
4. Secure handling of data: Currently, there aren't any obvious security flaws in this function as it just manipulates local data. However, if in future versions, it's dealing with sensitive data, proper precautions should be taken.

2.0.992 `o_task_list`

Contained in `lib/functions.d/o_opensvc-task-functions.sh`

Function signature: 756a613eff908ff7ef9c499e18525dbd12006b68220d054fed53c6f97867db18

2.0.993 Function overview

`o_task_list()` is a Bash function designed for service management. It queries the list of services using a local `services` variable. If a specific service name is provided, it checks for the existence of that service and returns an error message if the service does not exist. If no specific service is identified, it queries all existing services. Then, it extracts service details, retrieves a task list for each service, and iterates through each task, printing the task and its corresponding command.

2.0.994 Technical description

- **Name**: `o_task_list`
- **Description**: This function processes a list of services (or a single specified service), retrieving each service's configuration, extracting service details, and printing a list of tasks for each service.
- **Globals**: None.
- **Arguments**:
  - `$1` (optional): The name of a specific service to be processed. If no name is provided, the function will process all services.
- **Outputs**: Prints a list of tasks for each service.
- **Returns**:
  - Returns `1` if a specified service is not found.
  - Returns `0` when the processing and printing of tasks are successful.

- **Example usage**: `o_task_list "my_service"`

2.0.995 Quality and security recommendations

1. Add input validation to the `service_name` argument to ensure the function is working with expected data types and formats.
2. Consider restricting the function's permissions to prevent unauthorized access and potential exploitation.
3. Implement error handling to ensure that the function does not continue to execute if there is an error at any point in the process.
4. Use secure methods for handling and storing sensitive data (e.g., service configurations).
5. Use methods that are robust against common critical programming errors to improve the function's reliability and security.

2.0.996 `o_task_run`

Contained in `lib/functions.d/o_opensvc-task-functions.sh`

Function signature: a8188a18c4dff216f96650c63114ee4191457bbfc1912305cd1a643413523ed6

2.0.997 Function Overview

The `o_task_run` function is a Bash function that runs a specified task `task_id` on a specified service `service_name`. The function can be configured to apply to a specific node or all nodes. It validates the parameters, checks if the service and task exist, and if a specific node is requested, it validates if the instance exists. It then builds an `om` command to execute the task and logs the starting execution. After executing the task, it captures the output and logs the completion. In the case of failure, it registers errors accordingly.

2.0.998 Technical Description

- **name**: `o_task_run`
- **description**: A function to run specified tasks on specified services, possibly on a specific node.
- **globals**: [VAR: a variable used in the script]
- **arguments**:
  - `$1`: `service_name` - The name of the service on which to run the task.
  - `$2`: `task_id` - The ID of the specific task to run.
  - `$3`: `node` - The specific node on which to run the task. If left blank, the task runs on all nodes.
- **outputs**: Logs of task execution.

- **returns**: Returns codes indicating success or various types of errors.
- **example usage**: `o_task_run "service1" "task1" "node1"`

2.0.999 Quality and Security Recommendations

1. Make sure to use unique and identifiable names for your services, tasks, and nodes to make the function usage clear.
2. Always try to handle error situations gracefully, logging error messages that give sufficient information about what went wrong.
3. Take precautions to sanitize all input to protect from command injection or other malicious exploits.
4. Quality in scripting can be ensured by following good commenting practices to make the code understandable.
5. Avoid storing sensitive information like passwords in the script. Use secure ways to handle credentials.
6. Always validate the inputs before processing them.

2.0.1000 `o_vm_create`

Contained in `lib/functions.d/o_vm-functions.sh`

Function signature: 80ea2cc0b46e856dd51c246e33ff177372cbf92d050c7520cf1fa14fb9432edd

2.0.1001 Function overview

The function `o_vm_create()` is used to create a virtual machine (VM) on a specified target node. The function validates the provided VM identifier and target node, checks the state of the target node, logs the start of the operation, creates a service name, and then creates task service to build the VM. The function waits for instance availability on target node and once available, it executes the task. After the VM creation task completion, the function cleans up the task service and determines the correct return code based on the various scenarios of success and failure in the task execution and cleanup process.

2.0.1002 Technical description

- **Name:** `o_vm_create`
- **Description:** This function creates a virtual machine on a specified target node.
- **Globals:** None
- **Arguments:**
  - `$1`: vm_identifier: Identifier for the virtual machine to be created.
  - `$2`: target_node: Node on which the virtual machine is to be created.
- **Outputs:** Logs during various steps of the function.
- **Returns:** Based on the various conditions and scenarios:

  - Returns 1: If the parameters are invalid.
  - Returns 2: If the target node is not found in the cluster.
  - Returns 3: If the OpenSVC daemon is not running on the target node.
  - Returns 4: If the target node is frozen or in an error state.
  - Returns 5: If failed to create task service for VM.
  - Returns 6: If timeout occured while waiting for service instance on target node.
  - Returns 7: If Task failed but cleanup succeeded.
  - Returns 8: If Task succeeded but cleanup failed.
  - Returns 9: If both task execution and cleanup failed.
  - Returns 0: If both succeeded.
- **Example Usage:**

```
o_vm_create "vm1" "node1"
```

2.0.1003 Quality and security recommendations

1. Add data sanitization for parameters passed to the function.
2. Return specific error messages based on the error codes for better debugging.
3. Implement a timeout mechanism for each step for robustness.
4. Add additional error checking and handling for each function call within this script, especially handling for unknown errors.
5. Include logging for debugging and auditing purposes to track all activities related to VM creation.
6. The use of static values such as `max_wait=30` should be avoided. Consider replacing such instances with configurable parameters.
7. Regularly update and review your system for potential security holes + vulnerabilities.
8. Verify user permissions before allowing VM creation to avoid any unauthorized access or operations.

2.0.1004 `o_vm_get_healthy_nodes`

Contained in `lib/functions.d/o_vm-functions.sh`

Function signature: 0fd072e62375745d0379b61631516ed1a9edcdecc9c23774227390ad74f9b2d0

2.0.1005 Function Overview

The `o_vm_get_healthy_nodes` function takes a list of nodes as the argument and checks their health. The initial empty string and counters are set for keeping track of the healthy nodes and total count. The function loops through each node, validating its status, increments the total count, adds it to healthy nodes if it's valid, and adjusts the healthy count. The function then trims leading spaces in the end from the string of healthy nodes and logs the number of found healthy nodes out of the total. If any healthy

nodes are found, the function will display these nodes. The function will always return 0.

2.0.1006 Technical Description

- **Name:** `o_vm_get_healthy_nodes`
- **Description:** This function checks the health status of nodes passed in a list as an argument. It logs an error and returns 1 if the node list is empty and lists out the healthy nodes, if present.
- **Globals:** None
- **Arguments:** `[ $1: node_list ]` - A list of nodes passed as an argument.
- **Outputs:** Logs the number of 'healthy' nodes found out of the total and prints their list if present. Logs an error and returns 1 if the node list is not provided as an argument.
- **Returns:** Always returns 0, unless the node list is empty, in which case it returns 1.
- **Example Usage:**

```
nodes_list="Node1 Node2 Node3"

o_vm_get_healthy_nodes "$nodes_list"
# Output: Node1 Node3
# (Assuming Node1 and Node3 are healthy)
```

2.0.1007 Quality and Security Recommendations

1. Validate that each node in the node list is an acceptable format before checking the node's health, to prevent execution with improper data.
2. Consider improving error handling by ensuring the function exits upon encountering an error, instead of continuing to run through the loop.
3. Add more detailed logging messages to make debugging easier.
4. Your script should manually handle cases where a node is neither healthy nor unhealthy, like in a 'degrading' state.
5. Maintain a secure environment by making sure that sensitive information is not printed or logged and ensure log files have proper access permissions.

2.0.1008 `o_vm_validate_node`

Contained in `lib/functions.d/o_vm-functions.sh`

Function signature: 5e1059dd5c5fdd4d82e120d80acde2a5d59b54471914b6c5e96583c5e9f69c43

2.0.1009 Function overview

The `o_vm_validate_node` is a bash function designed to validate if a node is well suited for Virtual Machine operations. It checks on several parameters including the existence of the node in the cluster, node reachability and the frozen state of the node. Once validated, the function returns 0, otherwise, it logs the error and returns a non-zero integer.

2.0.1010 Technical description

- **name:** o_vm_validate_node
- **description:** This function validates a node for any VM operation by checking the node's existence, reachability, and frozen state.
- **globals:** Not applicable.
- **arguments:**
  - $1: The node_name that specifies the name of the node that needs to be validated.
- **outputs:** Logs information related to each of the checks performed, errors encountered if any, as well as whether the node validation was successful.
- **returns:** Returns 0 if all checks have been successfully passed. If not, it returns an error code (1, 2, 3 or 4) depending on where the function fails.

- **example usage:**

```
o_vm_validate_node node-1
```

2.0.1011 Quality and security recommendations

1. It is recommended to always provide the necessary argument (node_name) as failing to do so will result in an immediate error.
2. For improved visibility, consider breaking down the validation process into smaller functions, each handling one check. This makes the code cleaner and easier to debug.
3. Always check the return value of the function. If an error code is returned, appropriate measures need to be taken based on the type of error.
4. Ensure the grep command has secure and appropriate permissions, as it can be a potential security vulnerability.
5. Double check the JSON parsing code lines and ensure they're working as intended. This is because JSON parsing in shell scripting can be tricky and a small bug or typo could lead to unexpected behaviour.

6. Implement more robust error handling. For example, if a command fails to run it would be useful for the error output to provide more specifics for easier troubleshooting.

2.0.1012 `o_vm_validate_node_quiet`

Contained in `lib/functions.d/o_vm-functions.sh`

Function signature: 2af833f16f0c06eed232e207aa0d93907ba5ff03ef1888681985f81e9365f147

2.0.1013 Function overview

The function `o_vm_validate_node_quiet` is a bash function designed to validate a node in a certain cluster. This function takes a node name as its argument and applies a series of checks to verify its existence and reachability in the cluster as well as its 'frozen' state. The function returns with different exit codes following the completion of each check, allowing for more granular error detection.

2.0.1014 Technical description

- **Name:** o_vm_validate_node_quiet
- **Description:** Validates a using a series of checks digital node within a certain cluster.
- **Globals:** None
- **Arguments:**
  - $1: node_name - The name of the digital node to be validated
- **Outputs:**
  - Various exit codes (0 - 4) that correspond to validation success or the type of validation error encountered.
- **Returns:**
  - Returns 0 if node is successfully validated.
  - Returns 1 if the node_name parameter is missing or null.
  - Returns 2 if the node does not exist in the cluster.
  - Returns 3 if the node reachability check fails.
  - Returns 4 if the node is in a frozen state.
- **Example usage:**

```
o_vm_validate_node_quiet "node_name"
```

2.0.1015 Quality and security recommendations

1. Consider hardening the function against potential injection attacks, especially as it seems to handle user-supplied input.
2. As part of conciseness and DRY (Don't Repeat Yourself) principles, common code blocks that are used multiple times, such as the calls to retrieve 'status.gen' and 'frozen_at' data, could be extracted into separate functions.
3. Robust error handling could potentially be improved - currently, some significant failures such as issues invoking 'om' command or 'jq' are being piped to /dev/null.
4. Besides returning integer status codes, also consider providing descriptive error messages to better inform the user about the nature of an error.
5. Comments are helpful, consider adding some at critical parts of the code for maintainability.

2.0.1016 `parse_apkindex_package`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: c8272ffc02ed22252a554f027f250eccd763e445bdf8b52ad5784e2b6d338cbc

2.0.1017 Function overview

The Bash function `parse_apkindex_package` is designed to parse APKINDEX files. The function takes a package name and APKINDEX file as arguments, then finds and prints the paragraph associated with the package name. A paragraph ends with a blank line in APKINDEX file. If it does not end with a blank line, then it checks the last record. The function returns `0` if the package details are found or `1` if the package was not found.

2.0.1018 Technical description

- **Name**: `parse_apkindex_package`
- **Description**: This function is used to read an APKINDEX file, paragraph by paragraph (blank-line separated), and find package information. The package information is then echoed (printed) to the console. If the file does not end with a blank line, it checks the last record for matching package information.
- **Globals**: None
- **Arguments**:
  - `$1 (apkindex_file)`: The path of the APKINDEX file to be parsed.
  - `$2 (package_name)`: The name of the package for which information is sought.
- **Outputs**: Prints the information of the package found within the APKINDEX file.
- **Returns**: Returns `0` if the package details have been found in the APKINDEX and `1` if the package was not found.

- **Example Usage**:

  ```
  parse_apkindex_package "./apkindex.txt" "mypackage"
  ```

2.0.1019 Quality and security recommendations

1. Error Handling: The function currently doesn't handle possible exceptions. For instance, if the provided file is not accessible or does not exist.
2. Input Validation: There is currently no validation of inputs. This might lead to problematic behaviour if, for instance, an empty or incorrect APKINDEX filename or an empty package name is provided.
3. Documentation: The function could be clearer with more comments explaining any complex parts of the syntax used. Currently, there are no comments explaining what variables like `in_record` serve for, which might confuse other contributors or end users.
4. Efficiency: The function reads the file line by line, incurring heavy I/O operations. If the APKINDEX file is very large, this might not be efficient. There's room to optimize this for better performance.

2.0.1020 `parse_os_id`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: 377174d0352f5462a85cd347c6433a1bb72d67220d6e9555a200df9e2ed68095

2.0.1021 Function Overview

The `parse_os_id` function is designed to parse the unique identifier of an operating system. This identifier is passed as an argument to the function in the form of a colon-separated string. The function will then break down this string into three distinct parts - the OS architecture, the OS name and the OS version using the Internal Field Separator (IFS).

2.0.1022 Technical Description

- **Name:** parse_os_id
- **Description:** This function accepts a colon-separated string as input and breaks it down into three key components - the OS architecture, OS name, and OS version.
- **Globals:** None
- **Arguments:**
  1. `$1`: os_id - It is the colon-separated operating system identifier string.
- **Outputs:** This function does not output any value. It assigns values to OS_ID_ARCH, OS_ID_NAME, and OS_ID_VERSION based on the breakdown of os_id.
- **Returns:** N/A

- **Example Usage:**

```
os_id="amd64:ubuntu:18.04"
parse_os_id "$os_id"
echo $OS_ID_ARCH    # Output: amd64
echo $OS_ID_NAME    # Output: ubuntu
echo $OS_ID_VERSION # Output: 18.04
```

2.0.1023 Quality and Security Recommendations

1. Always make sure that you are passing a string in the appropriate format (architecture:name:version) for the function to work properly.
2. Ensure to validate the input and catch any errors if the input is not in the anticipated format.
3. Ensure that the function is compatible with the operating system's naming conventions.
4. Consider edge cases where the OS name or version contain unconventional characters that might break the parsing logic.
5. Avoid using global variables that may interfere with other functions or commands. Always use local variables unless a global variable is strictly necessary.
6. Always quote your variables to prevent word splitting.
7. Always check the return status of commands and functions used in your code to ensure they were successful.

2.0.1024 `prepare_custom_repo_for_distro`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: f41fadc94c0ebb53ebb87776324ccfcc73903f665dfe9bfd0a9d3f4c9cbb827c

2.0.1025 Function Overview

The function `prepare_custom_repo_for_distro` serves to prepare a custom software repository for a given linux distribution represented by the input `dist_string`. It also takes an array of URLs or local files to download and add to the custom repository. The function segregates URLs and local file paths from the other entries stamped as required packages in an iterative manner. Further, it initiates repository creation, downloads the package from each URL or copies from local files, and builds the YUM repository. It verifies that the required packages are available in the repository and logs an error if a package is missing.

2.0.1026 Technical Description

- **Name**: prepare_custom_repo_for_distro

- **Description**: This function prepares a custom software repository for a given distribution by segregating source links (URLs or local file paths) from required package listings, creating repositories, building YUM repository, and ensuring presence of required packages.
- **Globals**: *HPS_PACKAGES_DIR*: The root directory where repositories are created.
- **Arguments**:
  - `$1`: `dist_string`, the string representation of a Linux distribution.
  - `$@`: An array of URLs, local file paths to the required packages or names of the required packages.
- **Outputs**: Logs various informational and error messages during repository preparation. Copies or downloads package files to the repository.
- **Returns**: The function uses return values varying from 0-6 to indicate success or various types of failures (such as, failure to create repository directory, download or copy a package, etc.).
- **Example usage**: `prepare_custom_repo_for_distro "ubuntu" "https://example.com/package1.deb" "package2" "/path/to/package3.deb"`

2.0.1027 Quality and Security Recommendations

1. **Error Handling**: At present, the function returns error messages in various places but a more rigorous error handling system should be implemented for each step.
2. **Input Validation**: Currently, the repository name and the package sources are not validated. They should be confirmed to avoid unpredictable behavior.
3. **User Permissions**: Permissions required to create directories or copy files need careful considerations.
4. **Logging & Auditing**: It would be good to maintain consistent logging throughout the function to account for all actions taken.
5. **Dependency Handling**: The function should manage potential dependencies of the required packages.
6. **Use Secure Communication**: If possible, use secure protocols (like HTTPS) for downloading packages from URLs.

2.0.1028 `print_cluster_variables`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: c5583d8fd4715e19ec442b98a50c60b43c9b20d0f5892f8d147bbb29263d00fa

2.0.1029 Function Overview

The function `print_cluster_variables()` in Bash is used to read and print the variables from a configuration file of a currently active cluster. This function first checks if the config file exists, and if it doesn't, returns an error message and terminates with

a non-zero return value. If the config file exists, this function reads every line from it, ignores blank lines or lines starting with the hash symbol (#), and prints the rest after removing surrounding quotes.

2.0.1030 Technical Description

- **Name**: `print_cluster_variables()`
- **Description**: This function reads a configuration file for a currently active cluster and prints its variables after removing surrounding quotes. It returns an error and stops execution if the config file doesn't exist.
- **Globals**: No global variables are used.
- **Arguments**: This function doesn't take any arguments.
- **Outputs**: Variables from the cluster's config file. The output is sent to stdout.
- **Returns**: 1 if the config file doesn't exist. Otherwise, the return value is dependent on the final command in the function.
- **Example usage**: `print_cluster_variables`

2.0.1031 Quality and Security Recommendations

1. For enhanced security, consider adding additional checks besides the presence of the config file. For example, verifying file permissions or ownership.
2. Always quote variable references to prevent word splitting and globbing. For instance, consider replacing constructs like `$k` with `"$k"`.
3. The function could return a specific non-zero exit code for different types of failures (file not found, permission errors, etc.) to make error handling more specific.
4. To improve readability, consider adding a comment describing what each local variable specifically stores.
5. Consider handling possible errors in command substitutions used in variable assignments (like the call to `get_active_cluster_filename`) for more robust error handling.

2.0.1032 `_print_meta`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 8f66cc2d9e01b400ddc05a00d6399036188baa213e119c36c4340e95320bcf7f

2.0.1033 Function Overview

The function `_print_meta` is used to obtain and display metadata about a cluster configuration using a specific key and label. The function makes use of two arguments, the key and the label, to fetch the metadata details of a cluster configuration. It then prints the label and the value if the value is not empty. Furthermore, it also handles

conditions where there might be multiple clusters, none active cluster and also when there is no active one and exactly one cluster.

2.0.1034 Technical Description

- **Name:** `_print_meta`
- **Description:** A shell function to fetch and display metadata information from a cluster configuration.
- **Globals:**
  - `key`: The configuration key used to fetch value from the cluster configuration.
  - `label`: Label to be print before printing the configuration value.
- **Arguments:**
  - `$1`: The first argument, used as the key in the function.
  - `$2`: The second argument, used as the label in the function.
- **Outputs:** The function can output the label and configuration value if the value is not empty.
- **Returns:** The function returns 0.
- **Example usage:** `_print_meta "DESCRIPTION" "Description"`

2.0.1035 Quality and Security Recommendations

1. As per good practice, make sure that the necessary sanitization of input parameters is performed before they are used.
2. Error checking should be established to handle cases where the cluster configuration or key provided as function argument might not exist.
3. Clear readability should be maintained throughout the code. Usage of clear variable names and comments can help to ensure a good level of readability.
4. Always consider the security implications of your script; be cautious of the possibility of code injection attacks.
5. Regular updates and security patches should be implemented to ensure your bash shell is up-to-date and secure from known vulnerabilities.

2.0.1036 `process_host_audit`

Contained in `lib/functions.d/host-functions.sh`

Function signature: d86340d5a0f66f02b9b95c292f629ba33eb58698067514d3e83a0a1f1cb2d69a

2.0.1037 Function Overview

This is a Bash function named `process_host_audit()`. It takes three positional arguments, with the third being optional and defaulted to “host”. This function performs a host audit by decoding provided data, preparing and processing this data, and then storing the decoded data mapped with respective keys in a temporary file. It processes

the MAC address audit along with the associated data. Upon completion, it deletes the temporary file, stores some timestamp and count metadata, and logs the number of fields processed.

2.0.1038 Technical Description

- **Name:** process_host_audit
- **Description:** This function processes host audit by taking a MAC address, encoded data, and an optional prefix. It decodes the encoded data, prepares it, and stores it. After the successful completion of these operations, it erases the temporary file, stores the timestamp metadata and counter, and logs how many fields have been processed.
- **Globals:** [None]
- **Arguments:** [$1 - MAC address: A string representing the Media Access Control address, $2 - Encoded Data: A string of encoded data to process, $3 - Prefix: An optional string that prefixes the metadata entries]
- **Outputs:** This function logs the number of fields processed along with a message saying that host data collection is completed.
- **Returns:** The function returns 0 upon processing the function successfully indicating a successful completion of the function.
- **Example usage:**

```
process_host_audit "00:0a:95:9d:68:16"
↪  "%7B%22name%22%3A%22John%22%2C%22age%22%3A30%2C%22city%22%3A%22New%20York%22%7D"
```

2.0.1039 Quality and Security Recommendations

1. Consider using `mktemp` to generate a unique temporary file. This helps avoid file collisions and potential security risks by randomizing the filename.
2. Verify the input data for malicious inputs that could lead to code injection. Even though individual values within the loop are sanitized to check for invalid data, validating input before it enters the loop could offer an additional layer of security.
3. As the function relies on the existence of other functions (`ipxe_header`, `hps_log`, `host_config`), make sure that those functions perform their own error checking.
4. It may be beneficial to add error checking after operations that could potentially fail, such as file deletion or printf operations, to further improve the robustness of the function.

2.0.1040 `register_source_file`

Contained in `lib/functions.d/prepare-external-deps.sh`

Function signature: 3a47ee217d4c21d4f884f60289c0b97418f2bea9352f1117fe8517c7bb30bb0b

2.0.1041 Function Overview

The function `register_source_file()` has been designed to register a source file to an index within a specific destination directory. This function takes in a filename and handler as inputs and if any duplicate is avoided, it proceeds to register the file and its handler to the index file located in the destination directory. The function also provides feedback on successful registration or if the file is already registered.

2.0.1042 Technical Description

Name: `register_source_file`

Description: This Bash function registers a source file to an index present in the given destination directory.

Globals:
- HPS_PACKAGES_DIR: Path to the packages directory.

Arguments:
- `$1` or `filename`: The name of the source file to be registered. - `$2` or `handler`: The handler associated with the source file.

Outputs:
- Prints a message that indicates successful registration or if a source file is already registered.

Returns:
- Returns 0 if file is already registered, effectively preventing any changes.

Example Usage:

```
register_source_file "myfile.txt" "myHandler"
```

2.0.1043 Quality and Security Recommendations

1. Validate the inputs: Make sure both filename and handler are not empty or null before proceeding with the registration process.
2. Error handling: Add error catching mechanisms to handle unexpected situations such as issues with directory creation or file writing.
3. Secure Files: Ensure that the permissions for both the index file and directory are set appropriately to prevent unauthorized access.
4. Logging: Introduce detailed logging in each step for easier debugging and traceability.

2.0.1044 `remote_function_lib`

Contained in `lib/functions.d/kickstart-functions.sh`

Function signature: b6d9cd335e4f61b186f614aa07279b2f8bdd77298bf573a231ffd21869c18118

2.0.1045 1. Function Overview

The `remote_function_lib` function in Bash is a method to output a collection of functions that are supposed to be introduced into the pre and post sections of a script or command sequence. This function can be extremely useful in scenarios where repeated sections of code are needed across various parts of a script, allowing the user to maintain the DRY (Do not Repeat Yourself) principle. While the function currently outputs a predefined set of functions directly into the pre and post sections, future enhancements might permit the functions to be defined within an independent file and imported when needed.

2.0.1046 2. Technical Description

- **Name**: `remote_function_lib`
- **Description**: A Bash function that outputs multiple functions to be used in pre and post script/command sequences.
- **Globals**: None.
- **Arguments**: No explicit arguments.
- **Outputs**: A collection of functions that can be injected into the pre and post sections of a script or command sequence.
- **Returns**: The function does not explicitly return any value; it instead outputs multiple functions to be used elsewhere.
- **Example Usage**:

```
remote_function_lib
```

2.0.1047 3. Quality and Security Recommendations

1. To enhance code readability, consider moving the return functions into an independent file. This independent file can then be called within this function.
2. It would be better to include proper namings of the functions as comments to enhance readability.
3. For security purposes, consider adding input validations when the function starts handling command-line arguments.
4. Avoid exporting unnecessary global variables. They can conflict with variables from other parts of the scripts or be manipulated by unauthorized users.
5. Always use the latest version of Bash to get higher security and better new features from the latest updates.
6. Regularly run your code through static code analyzers (like ShellCheck) to catch potential bugs and security issues.

2.0.1048 `remote_function_lib`

Contained in `lib/functions.d/kickstart-functions.sh`

Function signature: b6d9cd335e4f61b186f614aa07279b2f8bdd77298bf573a231ffd21869c18118

2.0.1049 1. Function Overview

The `remote_function_lib` function is designed for assembling functions to be injected into pre and post sections of certain scripts. It does this by using a technique called “here document” or `heredoc`, denoted by the `<<EOF` syntax. This is generally used to reduce the need for invoking echo repeatedly or for creating a temporary file containing the lines of text. This function in particular does not do anything else yet and waits for the user to fill in the necessary details.

2.0.1050 2. Technical Description

- **Name**: `remote_function_lib`
- **Description**: This function is a placeholder for functions to be injected into pre and post sections of a script. It deploys a `heredoc` approach for adding lines of text.
- **Globals**: None
- **Arguments**: None
- **Outputs**: Outputs a block of text that is written between the EOF markers.
- **Returns**: Depending on the usage (not specified in the provided function details).
- **Example usage**:

```
source remote_function_lib
```

> **Note:** For actual usage, the function injected within this `heredoc` should be implemented first.

2.0.1051 3. Quality and Security Recommendations

1. Be mindful while using `heredocs`. If not properly utilized, sensitive data may be inadvertently written to a file and may remain there even after the script has finished.
2. As this function is passive and doesn’t perform any actions except print to standard output, ensure that the functionality it provides is actually needed in your script.
3. Consider using functions as external files for better organization, modularity, and error handling.
4. Always validate and sanitize input to functions; although this function doesn’t take any inputs, it’s a good general practice.
5. Include comments to thoroughly explain the details of the function, its inputs, outputs, and what it specifically does.
6. Also consider handling possible errors and return suitable error codes/messages for better debugging.

2.0.1052 `remote_log`

Contained in `lib/functions.d/kickstart-functions.sh`

Function signature: 1f586056ba1b573eed69d32639c3940c1c742ba73af4aae917a1d65d36c5367c

2.0.1053 Function overview

The `remote_log()` function is created to send log messages from a local machine to a remote gateway server. The function takes one argument, which is the message that needs to be logged. Before the message is sent, it is URL-encoded to ensure that it is correctly received by the remote server. The encoding is done in a loop, character by character. After the encoding, the message is sent to the remote server using a `curl` HTTP POST request.

2.0.1054 Technical description

Name: - `remote_log()`

Description: - This is a Bash function created to send log messages from a local machine to a remote gateway server. It takes one argument, the `message`, which needs to be logged. The function first URL-encodes the message and then sends it to the remote server via a POST request using `curl`.

Globals: - `VAR`: Not applicable - `macid`: The Mac id of the local machine - `HOST_GATEWAY`: The IP address or URL of the remote server

Arguments: - `$1`: The initial message that needs to be logged

Outputs: - Sends an HTTP POST request to the remote server with the URL-encoded message

Returns: - Null

Example Usage:

```
remote_log "This is a test log message"
```

2.0.1055 Quality and security recommendations

1. Input validation: Ensure that the variable `message` does not contain malicious commands or SQL injections.
2. Error handling: Handle exceptions to avoid function crashes e.g., if the remote server URL is incorrect or unreachable.
3. Logging: Include more logging within the function for debugging purposes.
4. Encryption: Consider encrypting the message content for confidentiality and data integrity.
5. Immutable globals: Since `macid` and `HOST_GATEWAY` are used as globals, careful handling is necessary as their value shouldn't be easily manipulated.

2.0.1056 `resolve_alpine_dependencies`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: 689d5ee3849151be4f6f15527c700df7d9c8156b9a6e4b876f2b82c3b22f9d9e

2.0.1057 Function overview

The `resolve_alpine_dependencies()` Bash function takes two arguments, `apkindex_file` and `package_name`. Its primary purpose is to parse an APKINDEX file corresponding to an Alpine Linux package repository, extract information relating to a specific package and its dependencies from the file, and print the details to standard output. Specifically, it extracts the package name, its version, and its associated dependencies, skipping over any shared library or file dependencies. It then recursively resolves and lists the dependencies for each of these package dependencies.

2.0.1058 Technical description

- **Name**: `resolve_alpine_dependencies`
- **Description**: This function reads an APKINDEX file and recursively prints a list of the input package name's dependencies. It checks if the APKINDEX file exists and whether the package exists in the APKINDEX. For each found package it extracts its name and version and if there are dependencies, it recursively calls itself to resolve these dependencies.
- **Globals**: ['hps_log: Function for logging']
- **Arguments**:
  - `$1: APKINDEX file path`
  - `$2: Alpine Linux package name`
- **Outputs**: Corresponding filename for each resolved dependency package along with error messages related to the package resolution process. Outputs will be printed on stdout.
- **Returns**: It returns '1' in case of errors, like if the APKINDEX file or the package doesn't exists. There's no successful return value ('0'), function output is meant to be captured from stdout.
- **Example usage**: `resolve_alpine_dependencies "/path/to/APKINDEX" "package_name"`

2.0.1059 Quality and security recommendations

1. Implement a method for throwing errors instead of returning '1' when any occur.
2. Add error handlers to check whether `$1` and `$2` have been provided when the function is called.
3. Extend error handling to deal with cases in which a dependency package is not found in the APKINDEX file.

4. Increase the robustness of the function by validating APKINDEX content format.
5. Add more logging messages to provide insight into the function's execution and progression, especially during the recursive traversal of dependencies.
6. Use a more efficient and error-proof method than `echo` and `grep` chains for extracting field value.
7. Implement a limit on recursive depth to avoid potential problems with cyclic dependencies.

2.0.1060 `rocky_latest_version`

Contained in `lib/functions.d/iso-functions.sh`

Function signature: f42e139a07133ae36c4e9783c1fbb144e1167e59537bd374c0b346f853c77a3d

2.0.1061 Function overview

The `rocky_latest_version` function is used to fetch the latest version number of the Rocky Linux distribution. It retrieves the listing of the Rocky Linux distribution repository using `curl`, filters, sorts, and echoes out the version number of the latest release.

2.0.1062 Technical description

- **name**: `rocky_latest_version`
- **description**: This function retrieves the HTML of the Rocky Linux distribution repository, filters out the version numbers using regular expressions, sorts them in reverse order, and echoes out the latest release version.
- **globals**:
  - `base_url`: the URL of the Rocky Linux distribution repository
  - `html`: stores the html data fetched from the `base_url`
  - `versions`: array holding the version numbers sorted in reverse order (`-Vr` option for "version sort")
- **arguments**: None
- **outputs**: The latest Rocky Linux version number.
- **returns**:
  - 1 if either fetching the HTML fails, or no version numbers could be extracted from the HTML
  - otherwise, does not explicitly return anything
- **example usage**:

```
latest_version=$(rocky_latest_version)
echo "The latest Rocky Linux version is ${latest_version}"
```

2.0.1063 Quality and security recommendations

1. Since this function heavily relies on the format of the HTML file, it may become brittle with changes to the website structure. It would be more reliable to use an official API, if available.
2. Error handling can be improved. Currently, the function returns 1 in case of either connection failure or when no versions are found in the HTML. Considering these two situations could be differentiated for better troubleshooting.
3. For security reasons, consider validating the HTML content before processing, as maliciously-crafted content might lead to unexpected behavior.
4. Add comments to give context to the regular expression used in the `grep` command. This will improve maintainability for developers not familiar with this pattern.

2.0.1064 `script_render_template`

Contained in `lib/functions.d/kickstart-functions.sh`

Function signature: 064ddcb49f3688c1b0ede8ce884eae36c9ac96f5117338bdce1f62d5c6960a67

2.0.1065 Function Overview

The `script_render_template()` function is designed to remap all variable placeholders (`@...@`) with their corresponding values (`${...}`). The function cycles through all the variables using `compgen -v`, assigns the value of the variable to a local variable, and then adds it to an array of values for awk. The awk utility then processes the array of values, replacing each placeholder in the original string with the corresponding variable value.

2.0.1066 Technical Description

- **Name**: `script_render_template`
- **Description**: This function is used to remap variable placeholders with their actual value. It does this using the awk utility and a for loop iterating over all variables in the scope.
- **Globals**: None
- **Arguments**: The function does not require any arguments. It acts on all variables in its scope.
- **Outputs**: The function outputs a string with all `@var@` placeholders replaced with their corresponding `${...}` values.
- **Returns**: The function does not have a return value.
- **Example Usage**: Assume that the following variables are already defined in the context:

```
script_name='MyScript'
script_version='1.0'
```

If we call `script_render_template` in a context where a template string like `'This is @script_name@ version @script_version@'` is present, the function will output the string `'This is MyScript version 1.0'`.

2.0.1067 Quality and Security Recommendations

1. Always make sure the substitution values (`${...}`) are securely obtained and sanitized to prevent command injection attacks.
2. Consider validating the variable names that `compgen -v` produces to ensure they adhere to expected patterns and rules.
3. Beware of potential performance issues if the function is used in a context with a large number of variables.
4. Handle errors and exceptions gracefully. For instance, what should happen if a placeholder variable does not exist?
5. Ensure that the function fits well within your specific use case, as its current implementation is very general and may not be suitable for more specific tasks.

2.0.1068 `select_cluster`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: d6887af36fa9b9a8372e021a7e7c69665b0d3eee270caacdddd223f760546494

2.0.1069 Function overview

`select_cluster` is a shell function designed to effectively manage multiple clusters within a shell environment. Depending on the argument passed, it returns either the directory or the name of the selected cluster. If the shell is non-interactive, it picks the active or first cluster depending on different conditions. If the shell is interactive, it prompts the user to select a cluster from a list of available clusters.

2.0.1070 Technical description

- **Name:** `select_cluster`
- **Description:** This function is used to manage multiple clusters in a shell environment. It either returns the directory or the name of the active or first available cluster, depending on specific conditions.
- **Globals:** `[ HPS_CLUSTER_CONFIG_BASE_DIR: Base directory for the cluster configuration ]`
- **Arguments:** `[ $1: Sets the return mode to 'name' if value is '--return=name' ]`
- **Outputs:** Prints either the directory or the name of the selected cluster.

- **Returns:** Returns 1 if no clusters are found. Returns 0 after successfully selecting a cluster in either interactive or non-interactive mode.
- **Example usage:** `bash select_cluster --return=name`

2.0.1071 Quality and security recommendations

1. For better script security, ensure all input data is validated and sanitized to mitigate the risk of injection attacks.
2. Always check return values from functions and handle any potential errors appropriately.
3. Make good use of local variables to limit the scope to the current shell and to avoid possible variable name conflicts.
4. Use double quotes around variable references to avoid word splitting and pathname expansion.
5. Regularly update the script and maintain proper documentation for easier debugging and maintenance.

2.0.1072 `select_network_interface`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: 019a6a43a2b588278bd5945964c48b92ffb331b67f1cacc18bedd6dd6828d661

2.0.1073 Function Overview

The function `select_network_interface()` is designed to present a selection menu to the user for choosing a network interface. The function creates a list of available network interfaces, and optionally adds a “None” option. Invalid selections are handled properly, and on a successful selection, the name of the selected interface is returned.

2.0.1074 Technical Description

- **Name**: `select_network_interface`
- **Description**: Shows a user-friendly selection menu to select one of the available network interfaces. Can optionally include a “None” option. Returns the name of the selected interface (not the full label shown in the menu).
- **Globals**: None
- **Arguments**:
  - `$1`: The prompt to be used in the selection menu (default: “Select network interface”).
  - `$2`: Flag to indicate whether to include “None” as an option (default: false).
  - `$3`: The display text for the “None” option, if included (default: “None”).
- **Outputs**:
  - If a valid interface is selected, its name is printed to stdout.

  - If “None” is selected, “NONE” is printed to stdout.
  - If an invalid selection is made, an error message is printed to stderr.
- **Returns**:
  - `0` if a valid selection is made.
  - `1` if the menu is exited without a valid selection.
- **Example Usage**: `selected_interface=$(select_network_interface "Choose an interface" true "No interface")`

2.0.1075 Quality and Security Recommendations

1. Always quote variable expansions and command substitutions to prevent issues with word-splitting and globbing. In this function, it is done correctly.
2. Consider checking if the `get_network_interfaces` command succeeded before proceeding, possibly exiting early if it failed.
3. Be cautious with using redirection (`< <(...)`), it can create a subshell and modify the parent shell’s state, which might lead to unexpected behavior.
4. Keep careful track of what you choose to expose to the user in the prompt. Depending on the context in which the function is used, some information might not be appropriate to share.
5. Check the inputs to the function to ensure they are as expected, and consider handling edge cases more explicitly. For example, what if the `include_none` argument is not a boolean?

2.0.1076 `set_active_cluster`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: ff67ed1fe94af1bc0a6ebd9436efd66e00f5b5f9905b2979af037bdf49fce60e

2.0.1077 Function Overview

This function `set_active_cluster` is responsible for defining the active Kubernetes cluster by name, in a specific environment. It accomplishes this through several steps - assigning the input to a variable, checking to see if the input variable is empty, defining cluster directory and configuration file locations, ensuring both the directory and configuration file exist, and finally, linking to the active cluster.

2.0.1078 Technical Description

- **Name**: `set_active_cluster`
- **Description**: This function sets a specific Kubernetes cluster as active based on cluster name provided as an argument.
- **Globals**: None
- **Arguments**:

  - `$1: cluster_name` - Name of the Kubernetes cluster to be set as active.
- **Outputs**:
  - Echoes an error message and usage suggestion to stderr if no argument is supplied, or if cluster directory or configuration file cannot be found.
  - Echoes a success message if the active cluster is successfully set.
- **Returns**:
  - Returns 1 if no argument is supplied or if cluster directory or configuration file cannot be found.
  - Returns 2 if clusted configuration not found.
- **Example usage**:

```
set_active_cluster my_cluster_name
```

2.0.1079 Quality and Security Recommendations

1. Input validation: Implement more comprehensive input validation; currently, the function only checks if an argument is supplied, but there might be specific naming rules for cluster names that could also be checked.
2. Error handling: More specific error messages could help with easier problem diagnostics.
3. Security: Review if and where this script allows for problems like symlink attacks; if a potential exists, steps should be introduced to minimize the risk.
4. Robustness: Consider introducing checks for edge cases such as file permission issues or lack of disk space.
5. Testing: Incorporate this function in unit testing to ensure it continues to work correctly as the cluster environment evolves.

2.0.1080 `_set_ips_hostname`

Contained in `lib/functions.d/dns-dhcp-functions.sh`

Function signature: 700b36c4d6e0fd66653208f81c915f9adc910edd5f1d6ccaeb6ab92fd6310776

2.0.1081 Function Overview

The bash function `_set_ips_hostname` sets the hostname of the device it is run on to a specified string. It sets `IPS_HOSTNAME` to “ips” and then uses the `hostname` command to change the machine hostname to “ips”. It further writes “ips” to `/etc/hostname`, which determines the system’s hostname.

2.0.1082 Technical Description

- **Name**: `_set_ips_hostname`
- **Description**: This function is responsible for setting the device’s hostname to “ips”. It assigns the name “ips” to `IPS_HOSTNAME`, changes the hostname using

the hostname command, and finally writes “ips” to /etc/hostname, which solidifies the hostname change.

- **Globals**: [IPS_HOSTNAME: This is a string variable that holds the new hostname to be set, in this case, “ips”]
- **Arguments**: This function takes no arguments.
- **Outputs**: This function does not explicitly produce any output, but changes the device’s hostname.
- **Returns**: This function doesn’t return any specific value, its main responsibility is to change the device’s hostname.
- **Example Usage**:

```
_set_ips_hostname  #sets hostname to "ips"
```

2.0.1083 Quality and Security Recommendations

1. The value “ips” is hardcoded into the function. Consider accepting this as a parameter so the function is more flexible.
2. Related to above, error checking should be done on any parameters passed to the function to ensure they’re safe and valid.
3. The function operates directly on /etc/hostname, which is a sensitive system file. Ensure necessary permission checks are in place before executing.
4. Any operation on system files could result in an unresponsive system if something fails. Consider implementing error handling and recovery methods.
5. The function does not provide output which can make debugging difficult. Consider adding optional verbose or debug mode to confirm the function’s operation.
6. Avoid having the function silently fail. If a step fails, the function should exit and report an error.
7. Ensure the script running the function has appropriate permissions to modify the hostname.

2.0.1084 _set_ips_hostname

Contained in lib/functions.d/dns-dhcp-functions.sh

Function signature: 700b36c4d6e0fd66653208f81c915f9adc910edd5f1d6ccaeb6ab92fd6310776

2.0.1085 Function Overview

The function _set_ips_hostname () is a bash function that is written to set the hostname of a system to “ips”. This function was designed to be used in Unix or Linux-based systems to modify the hostname for system identification on a network.

2.0.1086 Technical Description

- **Name**: _set_ips_hostname

- **Description**: This function changes the hostname of the system to the specified string “ips”. First, it declares a variable `IPS_HOSTNAME` and assigns the string “ips” to it. Then it uses the `hostname` command to change the system’s hostname, and finally writes the `IPS_HOSTNAME` value to `/etc/hostname`, effectively persisting the change across system reboots.
- **Globals**: `IPS_HOSTNAME`: Holds the hostname value to be set.
- **Arguments**: None.
- **Outputs**: Writes “ips” on standard output (through `echo` command) and to `/etc/hostname` file in the system.
- **Returns**: Returns the exit status of the last command executed in the function. Will return `0` if successful, or the error status code if the command fails.
- **Example Usage**: `_set_ips_hostname`

2.0.1087 Quality and Security Recommendations

1. The function does not currently handle any error conditions. For example, it will not check if we have permission to write to `/etc/hostname`. Therefore, adding an error handler to deal with such situations would increase the function’s robustness.
2. A hardcoded string “ips” is used as the hostname. In general, it is more flexible to allow this as an argument for the function which provides more use cases.
3. It’s important to ensure that writing to `/etc/hostname` file is secure. The function could be updated to run checks for any possible security vulnerabilities.
4. Lastly, improve function portability by checking the system type before executing - not all systems follow the same practice for setting hostnames, especially different Linux distros.

2.0.1088 `storage_get_host_vlan`

Contained in `lib/functions.d/storage_functions.sh`

Function signature: e63a3a1fba8ac0a209a0abe4bcc73383ffdfe3060c25018bef1faedec9aa1b77

2.0.1089 Function Overview

The function `storage_get_host_vlan()` is a bash function that takes in a MAC address as its argument and attempts to get the corresponding VLAN from the storage of the host. The MAC address is first normalized for correct formatting with the help of the `normalise_mac` function. If the normalization fails, the function returns an error.

If the normalization is successful, the function checks the host’s config for the VLAN corresponding to the normalized MAC address. If it finds a VLAN, it outputs the VLAN and ends successfully. If it does not find a VLAN, it returns an error.

2.0.1090 Technical Description

- **Name:** `storage_get_host_vlan`
- **Description:** This function gets the VLAN of a host's storage given a MAC address. It normalizes the MAC address and checks the host's config.
- **Globals:** None
- **Arguments:**
  - `$1`: The mac_address - The MAC address of the host
- **Outputs:** The function will output the VLAN of the host's storage or nothing if it does not retrieve it.
- **Returns:** The function returns `0` on success and `1` on failure.
- **Example usage:** `storage_get_host_vlan "00:0a:95:9d:68:16"`

2.0.1091 Quality and Security Recommendations

1. Validate the input: Ensure that the MAC input provided is a string and also in the right format.
2. Treat Global Variables as Read-Only: Although no globals are used in this function, as a good practice globals should ideally only be read and not written to. This will help avoid side effects.
3. Error Messages: The function could give a more detailed description when it returns 1 - failure. It may be important to know if it was the normalization process that failed or the retrieval of the VLAN from the host's config.
4. Naming Convention: Ensure that MAC address variable names clearly convey whether they are normalized or not (`normalized_mac_address` vs `mac_address`) to prevent confusion.
5. Test edge cases: Check the behavior of the function with invalid inputs, such as null values or special characters may need to be tested. This allows for the function to handle these appropriately.

2.0.1092 `storage_register_dns`

Contained in `lib/functions.d/network-functions.sh`

Function signature: f913020b7d6f0742b91ae844dd75de32721520d558f6468875a9fe6b4f606457

2.0.1093 Function Overview

The Bash function `storage_register_dns()` is used to register storage networks to domain name system (DNS). It gets clusters' hostnames, the count of storage networks, and the cluster domain. For each hostname, it retrieves its Media Access Control (MAC) address and iterates over the storage networks. If the IP address and the VLAN (Virtual Local Area Network) exist for a particular storage in the network, it registers them to DNS. If already registered, it will send a debugging message.

2.0.1094 Technical Description

Name: `storage_register_dns()`

Description: This function retrieves all hosts' hostnames, counts the storage networks and checks cluster domain in an infrastructure. Iterates over each hostname and for each storage network; it registers the IP and VLAN in DNS if they exist.

Globals: - `hostnames`: List of all hostnames in the cluster - `storage_count`: Number of storage networks in the cluster - `cluster_domain`: Domain used by the cluster

Arguments: - `$1`: Unknown, no arguments are explicitly used in the function - `$2`: Unknown, no arguments are explicitly used in the function

Outputs: - Registration status of DNS for each hostname and their storage networks. - Errors if no hosts are found to register or storage networks are not configured.

Returns: - `1` if no hosts are found to register or storage networks are not configured - `0` on successful completion

Example Usage:

```
storage_register_dns
```

2.0.1095 Quality and Security Recommendations

1. Input validation: Include checks for validity of hostnames, MAC addresses and IP addresses.
2. Error handling: Improve error handling by printing more descriptive messages that help in pinpointing issues.
3. Security: Ensure the function doesn't disclose sensitive details in its outputs like MAC addresses and IP addresses.
4. Logging: Add more detailed logging for tracking of operations, and for possible auditing purposes.
5. Comment code: Add comments to parts of the code to improve readability and maintainability.
6. Usage of function arguments ($1, $2): The function can be improved by using arguments for more versatility such as explicitly passing hostnames or storage networks to work with.

2.0.1096 `strip_quotes`

Contained in `lib/functions.d/network-functions.sh`

Function signature: b2892c33de60887f65c7f2b7946fafee0d873041d3a244be57b49c6339bb1cb5

2.0.1097 Function overview

The `strip_quotes` function is designed to process a string and remove leading and trailing quotes. This includes both single ('') and double ("") quotes. The function

achieves this through the use of local bash string manipulations.

2.0.1098 Technical description

- **Name:** `strip_quotes`
- **Description:** The function takes one string argument with either leading or trailing or both types of quotes and remove them.
- **Globals:** None
- **Arguments:**
  - `$1: str` This is a string input sent to the function. It is expected to have leading and/or trailing quotes which are to be removed.
- **Outputs:** The function prints the input string with the quotes removed.
- **Returns:** The function does not explicitly return a value, it only outputs the string without quotes via an echo command.
- **Example usage:** `strip_quotes "\"Hello World!\""` or `strip_quotes "'Hello World!'"`

2.0.1099 Quality and security recommendations

1. As the function does not account for inner quotes within the string, it is advisable to extend its functionality to handle such cases.
2. While executing scripts, always validate/filter the inputs for any malicious content, as appropriate. In this context, the function could be extended to check and validate the input string.
3. Since the function echoes the output, it might lead to risk of command injection attacks. A better way would be to return the value and let the caller decide what to do with the returned value.
4. Provide more descriptive error messages that would explain the problem if the script fails.
5. Increase resilience of the function by handling edge cases such as empty strings or strings without quotes.

2.0.1100 `_supervisor_append_once`

Contained in `lib/functions.d/configure-supervisor.sh`

Function signature: 1eae581e2531b97456c9424162821fe36555f053f09d123db7704d74e659d793

2.0.1101 Function Overview

The Bash function `_supervisor_append_once()` is used to configure the supervisor services. It takes a “stanza” - a service program identifier and a “block” of configuration instructions, and writes them to the `SUPERVISORD_CONF` file that governs supervisor settings. The function checks if the service already exists and only adds the

configuration if it's not present. If the writing operation fails, it returns an error code 3. It follows the same logic across multiple services: dnsmasq, nginx, fcgiwrap, rsyslogd, opensvc, and an event listener for post_start_config. At the end, it validates the existence and readability of the SUPERVISORD_CONF file.

2.0.1102 Technical Description

The `_supervisor_append_once()` function can be described as follows:

- Name: `_supervisor_append_once()`
- Description: A function that handles the addition of supervisor services configuration to a config file. It only appends configuration if the service doesn't already exist in the file.
- Globals: [SUPERVISORD_CONF : Global variable storing the path to the supervisor configuration file]
- Arguments:
  - `$1: stanza` - represents the identifier of the service.
  - `$2: block` - contains the configuration information for the service.
- Outputs: Debug and error messages to the log related to the status of addition or existence of supervisor services.
- Returns: It returns a status code 3 if the operation to append the configuration or validate supervisor configuration file fails, otherwise it returns 0.
- Example Usage:

```
_supervisor_append_once "program:dnsmasq" "$(cat <<EOF
[program:dnsmasq]
command=/usr/sbin/dnsmasq -k --conf-file=${DNSMASQ_CONF}
 ↪  --log-facility=${DNSMASQ_LOG_STDOUT}
autostart=true
autorestart=true
stdout_logfile=syslog
stderr_logfile=syslog
#stderr_logfile=${DNSMASQ_LOG_STDERR}
#stdout_logfile=${DNSMASQ_LOG_STDOUT}

EOF
)" || return 3
```

2.0.1103 Quality & Security Recommendations

1. This function lacks a mechanism to handle situations where the supervisor configuration file (SUPERVISORD_CONF) is not defined or misconfigured - for security and reliability, checks should be implemented.

2. To avoid a situation where overwritten global settings in the middle of a process result in undefined behaviour, consider making a local copy of all global variables used in the function.
3. To improve the quality of logging, consider adding more context in the error and debug messages. This could help in debugging and tracking operations.
4. Return codes should be documented and handled correctly by the caller to not allow the function to silently fail, enhancing robustness.
5. A more structured error handling could be implemented instead of directly returning from the function whenever an error occurs. This would enable error logging and cleanup operations.
6. Grasping the roles of "stanza" and "block" may be complex for a novice. Therefore, broader input validation and error messages could help in accurate debugging and improve user-friendliness.

2.0.1104 `supervisor_configure_core_config`

Contained in `lib/functions.d/configure-supervisor.sh`

Function signature: 6fc28f21a2369234dfffadccd828b971c8a610ef87ca5b9ea3782cafbe38792a

2.0.1105 Function Overview

The `supervisor_configure_core_config` function is responsible for setting up core configuration for the Supervisor process control system. Initially, this function validates that required environment variables are set. It then proceeds to create necessary directories for configuration and logging purposes, failing gracefully if it's not able to do that. The function writes the configuration for Supervisor into a specific file and validates that the file exists, can be read, has content, and includes critical configuration sections. Finally, the function returns a success message if all tasks were performed successfully.

2.0.1106 Technical Description

Name: `supervisor_configure_core_config`

Description: This bash function configures core Supervisord settings, including creating necessary config and log directories, and verifying the integrity of the configuration file.

- **Globals:**
  - `HPS_LOG_DIR` - Description: Base directory for logs; needs to be declared and exported in the environment.
- **Arguments:**
  - None
- **Outputs:**

  - Logs to HPS_LOG_DIR directory
  - Writes to a Supervisor configuration file

- **Returns:**
  - If the function executes successfully, it returns 0.
  - If environment variables are not set or cannot locate necessary directories, it returns 1.
  - If the function fails to create necessary directories, it returns 2.
  - If the function fails to write to the supervisor core configuration, it returns 3.
  - If the configuration file is not readable, does not exist after write, appears truncated or misses required sections, it returns 4.

- **Example usage:**

```
supervisor_configure_core_config
```

2.0.1107 Quality and Security Recommendations

1. The function should further validate that the paths it works with, such as for directories and configuration files, are well-formed and free of characters that raise security concerns (e.g., "../", "*","?")
2. In accordance to the principle of least privilege, the function should avoid running operations as root when not necessary.
3. A case where the configuration file could be overwriting an important existing file should be handled, to prevent data loss.
4. To support better debugging and audit, this function could log both successful and unsuccessful operations with appropriate log levels.
5. This function could take advantage of more bash built-ins to minimize reliance on external commands, potentially enhancing its speed and reliability.

2.0.1108 supervisor_configure_core_services

Contained in lib/functions.d/configure-supervisor.sh

Function signature: 265bb0e7d75175549cf4abf9fb58c0dc6e3ae913f4693c2ad20d6f25a0ec8721

2.0.1109 Function overview

The supervisor_configure_core_services function is used to prepare and confirm the core configuration file for Supervisor, a client/server system that allows users to monitor and control unix processes. The function employs all required directories, validates their existence, and logs respective messages for each process. The function will return error messages if paths to the core configuration or directories cannot be created.

2.0.1110 Technical description

- **Name:**
  - `supervisor_configure_core_services`
- **Description:**
  - The function that ensures the core Supervisord configuration exists, validates the configuration file, creates any directories required for Supervisord's operation and logs all required information.
- **Globals:**
  - `SUPERVISORD_CONF`: Path to the supervisor core configuration
  - `HPS_LOG_DIR`: Directory where log files are stored
- **Arguments:**
  - None
- **Outputs:**
  - Error and information logs regarding the supervisor configuration process.
- **Returns:**
  - 1: If retrieving the supervisor configuration path fails or if the configuration file is not found
  - 2: If directory creation fails
- **Example Usage:**
  - `supervisor_configure_core_services`

2.0.1111 Quality and Security recommendations

1. Implement more specific error handling: currently, the function simply returns an error code without further elaboration on the type of encountered issue.
2. Include input sanitation to the function, even if it currently does not take arguments, in preparation for future development.
3. Leveraging a version control system could help manage changes to the supervisor core configuration file, allowing easy rollback to previous versions if and issue arises.
4. Securely handle creation and permissions of log and configuration directories to prevent unauthorized access.

2.0.1112 `_supervisor_post_start`

Contained in `lib/functions.d/supervisor-functions.sh`

Function signature: 993b095b0f840da698a8e8e207eb57f863d1852d1d6f96728a61408d72c744a6

2.0.1113 Function overview

The function `_supervisor_post_start` is a Bash function that appears to be part of a larger orchestration and management system. Its purpose is chiefly to conduct some

form of configuration for a cluster system, presumably within the confines of the parent system or service name.

2.0.1114 Technical description

This summary details the `_supervisor_post_start` function with a focus on its technical specifications and characteristics.

- **Name**: `_supervisor_post_start`
- **Description**: A Bash function that is designed to handle configuration for cluster systems.
- **Globals**: [osvc_configure_cluster]
- **Arguments**: None. There are no arguments ($1, $2) identified.
- **Outputs**: Not determinable from the current single line. The function calls `osvc_configure_cluster` and any output would be defined by that function.
- **Returns**: No return statement is present in the function
- **Example usage**: `_supervisor_post_start`

2.0.1115 Quality and security recommendations

To improve code security and quality, the following might be worthy considerations:

1. If feasible, adding comments or a brief documentation snippet to the function would make understanding and maintenance more straightforward for others.
2. The function is currently lacking any error handling. Adding some form of error checking could make the execution much more robust, particularly in cases where `osvc_configure_cluster` fails to execute properly.
3. Lastly, assuming `osvc_configure_cluster` is a global function, verify that the function and its corresponding variables/data are properly scoped and secured.

2.0.1116 `_supervisor_post_start`

Contained in `lib/functions.d/supervisor-functions.sh`

Function signature: 993b095b0f840da698a8e8e207eb57f863d1852d1d6f96728a61408d72c744a6

2.0.1117 Function overview

This Bash function, `_supervisor_post_start`, is designed to configure a service cluster in a system. Its primary job is to invoke another function `osvc_configure_cluster` which is assumed to carry out the necessary configuration steps for a service cluster.

2.0.1118 Technical description

Name: `_supervisor_post_start`

Description: This function calls the `osvc_configure_cluster` function. It does not consume any argument nor alters any global variables.

Globals: None

Arguments: None

Outputs: The function does not have any specific output on STDOUT/STDERR. The output depends on the `osvc_configure_cluster` function it calls.

Returns: The return value of this function will be the same as the return value of `osvc_configure_cluster`.

Example usage:

```
_supervisor_post_start
```

Note: Since it uses an implicitly defined function `osvc_configure_cluster`, ensure that function is properly defined and behaves as expected in the environment before using `_supervisor_post_start`.

2.0.1119 Quality and security recommendations

1. **Explicit declaration**: Even though Bash allows using functions before their definition, it is recommendable to declare all functions explicitly before their usage for more readability and avoid potential bugs.
2. **Error handling**: Implement error handling and logging mechanisms within the `osvc_configure_cluster` function to capture and resolve faults.
3. **Environment Isolation**: Ensure that the function `osvc_configure_cluster` is working in the correct directory and using the correct user permissions, to protect against resource manipulation and privilege escalation.
4. **Data validation**: While this function doesn't have arguments, It is advisable to make sure that the `osvc_configure_cluster` function is validating its inputs properly, to mitigate any risk of code injection.
5. **Documentation**: Write clear documentation for the `osvc_configure_cluster` function including its inputs, outputs, side-effects, and return value, as `_supervisor_post_start` is entirely dependent on it.

2.0.1120 `supervisor_prepare_services`

Contained in `lib/functions.d/supervisor-functions.sh`

Function signature: b843c07babf2b3b72bacdd545d1136599a3bc427e4bf88d8bb1a3cb5801b8f04

2.0.1121 1. Function Overview

The `supervisor_prepare_services` function is a behind-the-scenes helper function in the Bash shell script that is used to prepare and configure various system services in proper order before the intended services can be run. It calls several other functions that manage important configurations in the system such as setting the hostname IP addresses, creating configurations for nginx, dnsmasq, rsyslog, and preparing the cluster for osvc.

2.0.1122 2. Technical Description

- **Name:** `supervisor_prepare_services`
- **Description:** This function helps to prepare system services by calling other functions that handle various configuration tasks. Functions called are: `_set_ips_hostname`, `create_config_nginx`, `create_config_dnsmasq`, `update_dns_dhcp_files`, `create_config_rsyslog`, and `osvc_prepare_cluster`.
- **Globals:** None.
- **Arguments:** No arguments are expected by this function - `supervisor_prepare_services`.
- **Outputs:** The function does not directly output anything. However, the sub-functions it calls might do so.
- **Returns:** Not explicitly defined in the function.
- **Example usage:** Calling the function without any arguments, like so - `supervisor_prepare_services`

2.0.1123 3. Quality and Security Recommendations

1. Since this function directly prepares system services, it holds a lot of power and permissions. It is good to review the permissions assigned to it and ensure only appropriate roles can call it.
2. Always ensure that only valid configurations are being passed to the various functions this function calls. Configurations should be double-checked to not contain any malicious code or data.
3. It is recommended to add error handling. At each step of the provisioning process, consider checking for the success of the previous step and handle errors or exceptions accordingly.
4. Verify and sanitize output of the sub-functions it calls to ensure they do not pose a security risk.
5. Perform regular audits of the script that this function is part of to identify any vulnerabilities or bugs that might be present.
6. While this function does not have any explicit arguments, it is unclear whether the sub-functions being called within have dependencies on global variables or other resources. Care should be taken to validate and sanitize all such dependent inputs.

2.0.1124 supervisor_prepare_services

Contained in `lib/functions.d/supervisor-functions.sh`

Function signature: b843c07babf2b3b72bacdd545d1136599a3bc427e4bf88d8bb1a3cb5801b8f04

2.0.1125 Function overview

The function `supervisor_prepare_services` serves to prepare some services on a computer running a Unix-like operating system. It includes setting the IP hostname, configuring Nginx, configuring Dnsmasq, updating DNS and DHCP files, configuring Rsyslog, and preparing the cluster for Open Service Catalog Manager (OSCM).

2.0.1126 Technical description

Name: `supervisor_prepare_services`

Description: The function prepares a set of system services in the following sequence: 1. Sets up the IP hostname
2. Creates a configuration for Nginx 3. Creates a configuration for Dnsmasq 4. Updates DNS and DHCP files 5. Creates a configuration for Rsyslog 6. Prepares a cluster for OSCM (Open Service Catalog Manager)

Globals: N/A

Arguments: No arguments are needed to execute this function.

Outputs: The function calls other functions that may have their own outputs. This function does not produce outputs on its own.

Returns: Does not return a value.

Example Usage:

```
# Call the function
supervisor_prepare_services
```

2.0.1127 Quality and security recommendations

1. Ensure proper input validation and error checking. Even though this function currently doesn't take any arguments, the functions it's calling may do. Proper input validation and sanitization should always be done to prevent potential command injections.
2. Limit permissions and privilege escalation. Shell scripts should be run with the minimum privileges necessary to complete the job to limit potential damage in the event of a bug or a security breach.
3. Logging: A commendable practice would be to introduce logging within this function to assist in tracking its execution.

4. Code readability: It would be beneficial if each function being called had a brief comment indicating its purpose and potential effects. This would enhance readability and maintainability, as other developers could understand the code more effectively.

2.0.1128 `_supervisor_pre_start`

Contained in `lib/functions.d/supervisor-functions.sh`

Function signature: 9002f106a8806de5485c032dc77ff9acfe11fb5eb5b1c302a2bdba54a29fc26f

2.0.1129 Function overview

The `_supervisor_pre_start` function is a bash function that is responsible for the preparation and configuration of supervisor core and services before start-up. It executes a series of other functions that configure the core settings, prepare the services, and reloads the core configuration.

2.0.1130 Technical description

- **Name:** `_supervisor_pre_start`
- **Description:** This function is employed in the start-up process, specifically before the supervisor starts. It ensures that the core configuration, core services, and services are all properly set and ready to be activated. It also reloads the core config file to apply all changes. The function does not expect any input or produces any output and directly applies changes.
- **Globals:** None.
- **Arguments:** None.
- **Outputs:** None directly. All changes are internal within the supervisor.
- **Returns:** None.
- **Example usage:**

```
_supervisor_pre_start
```

2.0.1131 Quality and security recommendations

1. Ensure all functions called within this function handle errors and exceptions appropriately. There should be a system in place to stop the start-up process if any critical configuration fails.
2. As a best practice, always validate and sanitize any input to the function calls inside `_supervisor_pre_start`. Even if the function does not take arguments directly, the functions it calls might.
3. Make sure to bind the actions of this function to the appropriate permissions, as it deals with configurations which if mishandled, can have significant effects.

4. Implement logging of activities within this function. This will help track and troubleshoot any issues that may arise during the start-up process.
5. Keep the architecture modular to allow for easy troubleshooting, debugging, and scalability. It's best to write functions that do one thing and do it well. In this case, each function called in `_supervisor_pre_start` should be responsible for only a specific component of the supervisor configuration.
6. Regularly update the supervisor versions and perform regular security audits to identify and rectify any potential threats.
7. Write comprehensive unit tests for these type of critical functions to ensure they are working as expected under all possible scenarios.

2.0.1132 `_supervisor_pre_start`

Contained in `lib/functions.d/supervisor-functions.sh`

Function signature: 9002f106a8806de5485c032dc77ff9acfe11fb5eb5b1c302a2bdba54a29fc26f

2.0.1133 Function Overview

The function `_supervisor_pre_start()` is a shell script function that consists of a sequence of calls to other functions to help manage pre-start setups in a Supervisor environment. The function initializes substantial attributes pertinent to the core configuration and services, prepares the services, and finally, reloads the core configurations in order to effect the changes made.

2.0.1134 Technical Description

Name: _supervisor_pre_start

Description: This function is part of a larger Supervisor system and is responsible for configuring core services and their configurations, preparing services, and reloading the core configs to commit changes in a Supervisor setup.

Globals: None in this function.

Arguments: None in this function.

Outputs: The function outputs are dependent on the embedded function calls. No explicit output within this function.

Returns: The function's return is also dependent on the embedded function calls. No explicit return within this function.

Example Usage: `_supervisor_pre_start`

2.0.1135 Quality and Security Recommendations

1. When employing this function, ensure that sufficient error handling and logging measures are in place. This way, if one of the function calls within `_supervisor_pre_start` should fail, there is an efficient way to capture the error and handle it appropriately.
2. It is highly recommended to restrict the permissions to this function to required users or processes. Since this function manipulates core configurations of services, unrestricted access could potentially harm the system.
3. Always test this function in a controlled environment before porting it to the production mode. Owing to the function's potential to change system settings, a faulty function execution could lead to severe disruptions.
4. Since this function does not utilize any arguments or global variables, ensure the called functions are well implemented with their own security precautions to prevent against any vulnerability.

2.0.1136 `supervisor_reload_core_config`

Contained in `lib/functions.d/supervisor-functions.sh`

Function signature: 908b803c09dae08b0c269eda962224cbee23deb1e098eefe33f75098c3aa5c80

2.0.1137 Function overview

The `supervisor_reload_core_config()` function in bash is used to reload the configuration of the Supervisor core. This function fetches the path to the Supervisor configuration, then logs and executes two commands: `reread` and `update`. Both of these commands are executed via `supervisorctl` with the configuration file as the argument. By re-reading and updating, any changes made to the configuration file are applied without the need to restart the Supervisor service.

2.0.1138 Technical description

- *Name*: `supervisor_reload_core_config()`
- *Description*: This bash function fetches the path to the Supervisor configuration, then logs and executes `reread` and `update` commands using `supervisorctl`.
- *Globals*:
  - SUPERVISORD_CONF: Points to the path where the Supervisor configuration file is located.
- *Arguments*: None
- *Outputs*: Logs the output of the `supervisorctl reread` and `update` commands.

- *Returns*: Nothing. It performs actions and output logs but does not provide a return value.
- *Example usage*: `supervisor_reload_core_config` ### Quality and security recommendations

1. Always ensure that the `supervisorctl` utility is available in the system and is up to date for correct functioning of this bash function.
2. Make sure the function has correct permissions set and can access the required `SUPERVISORD_CONF` path. This is essential for the function to perform correctly without any permission-related issues.
3. Add error handling for the function. Currently, it does not handle any potential errors that might occur while getting the Supervisor configuration path or executing `supervisorctl` commands.
4. In case of sensitive data printed in logs, make sure the logs are protected and securely stored.
5. Regularly review the logged data and function performance to detect any potential function- or configuration-related issues.

2.0.1139 `supervisor_reload_core_config`

Contained in `lib/functions.d/supervisor-functions.sh`

Function signature: 908b803c09dae08b0c269eda962224cbee23deb1e098eefe33f75098c3aa5c80

2.0.1140 Function Overview

The function `supervisor_reload_core_config()` is a bash shell function tailored for supervisor management. Supervisor is a process control system that allows its users to monitor and control a number of processes on UNIX-like operating systems. This function re-reads the supervisor configuration file and updates it.

2.0.1141 Technical Description

Function name:

supervisor_reload_core_config

Description:

The function fetches the path to the supervisord configuration file, then performs a reread and update action. This is done in coordination with the supervisorctl tool that communicates with supervisord to manage the processes.

Globals:

None

Arguments:

None

Outputs:

Logging information about the reread and update of the supervisor's configuration file

Returns:

No explicit return value

Example usage:

```
supervisor_reload_core_config
```

2.0.1142 Quality and Security Recommendations

1. It's a good practice to handle error exceptions, to manage potential failure in locating the supervisorctl or configuration file.
2. Make sure permissions on supervisor's configuration file restrict unauthorized access, ensuring that only the appropriate users can read, write and execute.
3. Logging mechanisms can be enhanced. Instead of just logging actions, try to log the status or any error messages during the process.
4. Avoid storing sensitive information in logs to prevent potential security risks.
5. Consider adding a mechanism for validating the success of the updated configuration.

2.0.1143 `supervisor_reload_services`

Contained in `lib/functions.d/supervisor-functions.sh`

Function signature: f7f5dbec5d6c487b41efa7323a44d1ec6d88cb0befd10f9de4bdfd11727c57bf

2.0.1144 Function overview

The function `supervisor_reload_services()` is designed to send a HUP signal (Hang UP) to services managed by `supervisord`. The function primarily verifies if the supervisord configuration file exists and then decides the target of the HUP signal. It logs the outcome of executing the HUP signal.

2.0.1145 Technical description

- **Name:** `supervisor_reload_services()'`
- **Description:** This function sends a HUP signal to either a specific `supervisord` service or all `supervisord` services, based on the provided arguments.
- **Globals:** `SUPERVISORD_CONF: The path to the supervisord configuration file.`

- **Arguments:** `$1: The name of the service to signal. If this argument is not provided, the function sends a HUP signal to all supervisor services.`
- **Outputs:** Logs an error message if the SUPERVISORD_CONF is empty or if the configuration file doesn't exist. It also logs the results of sending the HUP signal.
- **Returns:** Returns 0 if the HUP signal was sent successfully, returns 1 if the SUPERVISORD_CONF is empty or if the configuration file doesn't exist, and returns 2 if failure in sending the HUP signal.
- **Example usage:**

```
supervisor_reload_services my_service
```

2.0.1146 Quality and security recommendations

1. It's recommended to validate the input for `service_name`, ensuring it does not contain harmful characters that could lead to command injection vulnerabilities.
2. Logging should be correctly implemented to help troubleshooting and to avoid leaking sensitive information in case of errors.
3. Instead of returning numeric values, consider creating an enumeration of error types and return these for better readability.
4. Always consider edge cases where the SUPERVISORD_CONF is incorrectly set or the configuration file does not exist. This function is already handling these cases gracefully.
5. Caution should be taken into account when dealing with highly privileged services to avoid violating the principle of least privilege.

2.0.1147 `sync_alpine_packages`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: 6bb558f1bb3c0d8f2bfa22a90d4d17b85992b271c923c67052d9303ae5abe7c9

2.0.1148 Function Overview

The function `sync_alpine_packages` synchronizes packages from the Alpine Linux package repository. Using the Alpine version, mirror version, and repository name provided as input arguments, along with a list of package names, it creates a directory structure, downloads the APKINDEX, and attempts to extract it. If successful, the function iterates through the package names, resolves dependencies, removes duplicates, and downloads each package. The APKINDEX is copied into the created directory structure. The function then cleans up by removing the temporary directory and returns a status notification indicating successful synchronization or an error if any process fails.

2.0.1149 Technical Description

Name: sync_alpine_packages

Description: This Bash function synchronizes packages from the Alpine Linux package repository. It calls on several other functions in the process, including `_get_distro_dir`, `hps_log`, `resolve_alpine_dependencies`.

Globals: None.

Arguments: - `$1: alpine_version` - The version of Alpine Linux for which packages are sought. - `$2: mirror_version` - The version of the mirror from which packages are downloaded. - `$3: repo_name` - The name of the repository housing the desired packages. - `$package_names` - An array of package names to be downloaded.

Outputs: - Sync logs and error messages. - Downloads packages and their dependencies into a created directory structure. - Deletes the temporary directory created during operation.

Returns: - 2 when an error occurs (e.g., failed to create directory, failed to download or extract APKINDEX, failed to resolve dependencies). - 0 indicates successful syncing of packages and dependencies.

Example Usage:

```
sync_alpine_packages "3.12" "v3.12" "main" "gcc" "libc-dev"
```

2.0.1150 Quality and Security Recommendations

1. Consider accepting package names and other inputs through a secured method to avoid command injections.
2. Instead of using`wget`, consider using `curl` since it has more features and better handling for various situations.
3. Implement more robust log handling for both success and error cases, including more detailed information about the error (e.g., why download or extraction failed).
4. Handle failure of `wget` during packages download loop.
5. Clean temporary directory `$temp_dir` even if a function returns prematurely.
6. Avoid using explicit paths, and consider using variables instead for better maintainability.
7. Consider checking if the necessary programs (`wget`, `tar`, etc.) are installed before trying to use them.

2.0.1151 `sync_alpine_repo_arch`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: ff204afc2f7f40dd88298ae0ef2194944bab69755c11a6e2b90d00e3785cd96c

2.0.1152 Function overview

The `sync_alpine_repo_arch` function is intended to synchronize an Alpine Linux repository to a local directory. It takes in four parameters - alpine version, mirror version, repository name, and architecture (defaults to 'x86_64' if not provided). The function creates the destination directory, downloads repository files from the specified HTTP mirror, checks available space before each download, and cleans up any partial download in case of insufficient disk space.

2.0.1153 Technical description

- **Name:** `sync_alpine_repo_arch`
- **Description:** This function syncs an Alpine Linux repository to a local directory, checking available space before each download and performing cleanups as necessary.
- **Globals:** Not used in this function.
- **Arguments:**
  - `$1`: The alpine version.
  - `$2`: The mirror version.
  - `$3`: The repository name.
  - `$4`: The architecture type; defaults to 'x86_64' if not provided.
- **Outputs:** The function outputs the number of files found in the repository, logging every 20 downloads along with the available disk space, and the total number of files downloaded.
- **Returns:** Upon successful execution, the function returns the result of `validate_apkindex` call. In case of an error in creating the destination directory or insufficient disk space, the function exits with code 2.
- **Example usage:** `sync_alpine_repo_arch v3.9 latest main x86`

2.0.1154 Quality and security recommendations

1. Use more robust error handling: Currently, the function exits with a return code, but it might be worthwhile to throw exceptions or use a more comprehensive system for signaling error conditions.
2. Enhance logging: Include further details in the event of an error, such as the mirror version and architecture type on disk space check failure.
3. Use secure methods to download files: Instead of `wget`, switch to a command that verifies the SSL certificate of the server.
4. Regularly update the alpine and mirror versions: Ensures the function works with the latest security patches and updates.
5. Validate downloaded files: Add checksum verification for downloaded files.
6. Improve disk space check: Increase the frequency of disk space checks or even include a projected space requirements calculation.

7. Limit download retries: To prevent excessive internet usage, limit the number of retries for file downloads.
8. Clean up temporary files: Ensure that all temporary files are cleaned up in all potential exit points of the function to avoid littering the file system with temporary files.

2.0.1155 `tch_apkovol_create`

Contained in `lib/functions.d/alpine-tch-build.sh`

Function signature: 491c12dc2ecb4d137dc88e9a618080ecd29588cf589fbd64ee74f4affec3d658

2.0.1156 Function overview

The `tch_apkovol_create` function in Bash is used to produce Alpine apkovl (Alpine package formats) files. The function accepts two arguments: the output file and the ID of the operating system to be used. Validation is made to ensure the existence of the provided os_id, where upon failure, an error message is logged and the function terminates. The function then retrieves configuration details from the os_config and the cluster_config. If the necessary information isn't found or configured, the function will either log an error/warning and terminate, or use default values before proceeding. A temporary workspace is created for the building of the package, where components are built. If any step of the component creation fails, an error is logged, temporary files are cleaned up and the function terminates.

2.0.1157 Technical description

- **Name:** `tch_apkovol_create`
- **Description:** Creates an Alpine Linux bootstrap package file specified by the `os_id` and saves it to the specified `output_file`.
- **Globals:** None.
- **Arguments:**
  - `$1: output_file` Destination for the packaged tarball.
  - `$2: os_id` The ID of the OS to be used.
- **Outputs:** It logs various informational, error, or warning messages in the process.
- **Returns:** It returns `1` in case of validation or function execution errors and `0` if the apkovl was successfully created.
- **Example usage:** `tch_apkovol_create /path/to/output.tar.gz ubuntu`

2.0.1158 Quality and security recommendations

1. Defensive programming is recommended by providing default values for crucial variables and more thorough error handling.

2. Specific checks for existence of required files or directories before trying to access them can improve reliability.
3. Sanitizing inputs, particularly os_id, can help prevent potential security risks.
4. Using more specific error return values can help identify which step of the process failed.
5. It's recommended to document the function more thoroughly, especially the flags and arguments accepted by the function. Markdown formatting can be used to improve readability.

2.0.1159 `ui_clear_screen`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 0493ca3490c6e95475fb08d2f387298f827857616ec67718e0dd7bc3268a31d2

2.0.1160 Function Overview

The `ui_clear_screen()` function is a simple Bash utility function used to clear the terminal screen. If the `clear` command is available and successful, it will be used. Otherwise, it will fall back to outputting an escape sequence that should also perform the same function, this is especially handy in various system environments where the `clear` command may not be available.

2.0.1161 Technical Description

- Name: `ui_clear_screen`
- Description: This function is responsible for clearing the terminal screen. It first tries to use the `clear` command and falls back to an escape sequence (`\033c`) if `clear` is not successful.
- Globals: Not applicable as this function does not use or modify any global variables.
- Arguments: Not applicable as this function does not take any arguments.
- Outputs: A cleared terminal screen.
- Returns: If the `clear` command is successful, this function will return the exit status of the `clear` command which is expected to be 0 indicating success. If the `clear` command fails (non-zero exit status), 0 will be returned after printing the escape sequence.
- Example Usage: `bash ui_clear_screen()`

2.0.1162 Quality and Security Recommendations

1. Always ensure that functions correctly handle unexpected or erroneous input. For `ui_clear_screen()`, that's not necessary as it doesn't take any arguments.

2. The function doesn't provide or log any error messages which might be useful in some scenarios to debug issues related to the terminal or environment. You could consider adding error logging.
3. Evaluate the fallback method of using an escape sequence for screen clearing, it might not work or could lead to unexpected results in certain terminal environments. A more robust way of handling the unlikely event of `clear` failing could be devised.
4. Make sure to check the return values of commands you are running within your functions, incorporate error checking mechanisms wherever possible.
5. Consider outputting a warning or notice to the user when the fallback method is employed letting them know `clear` command wasn't successful.
6. This function should be safe from code injection as it does not process external input or environmental variables.

2.0.1163 `__ui_log`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: d3d49681e2b691a5ff908ab2cfc80feb43c6f2c7f2aa6c60521377957fc9244b

2.0.1164 Function Overview

The function `__ui_log` in Bash is a utility for logging or displaying messages with a distinctive prepend label. It can be utilized to show system-level or user interface related informational, warning, or error messages. The function outputs the passed messages to the standard error stream.

2.0.1165 Technical Description

- **Name:** `__ui_log`
- **Description:** This is a logging function used to display messages with a distinctive label '[UI]'. It's mainly intended for outputting system or user interface messages. The function employs the `echo` command to output the messages, which are passed in as arguments.
- **Globals:** None
- **Arguments:** `$*` - A list of arguments to be logged or displayed. They're concatenated into a single string by the `echo` command.
- **Outputs:** The function redirects its output to the standard error output stream (>&2). Hence, any message passed to this function would appear in the stderr stream, with '[UI]' as a prefix.
- **Returns:** None. The function doesn't explicitly return a value.
- **Example Usage:**

```
# Example to log a message
__ui_log "This is a UI log message"
```

2.0.1166 Quality and Security Recommendations

1. For added clarity and readability, it could be beneficial to add comments within the function to explain what each component does.
2. The function might be enhanced with the addition of explicit return values, aiding in error handling and flow control in scripts using this function.
3. To prevent command injection attacks, ensure that all variable data is properly escaped before it is included in the log message.
4. Implement a mechanism to control the log level. Thereby, control what kind of messages (debug, info, warning, error) should be directed to the output.
5. Always make sure that no sensitive data is logged in order to maintain information security and privacy.
6. Consider directing the logs to a specific file or a log management system, for easier troubleshooting and better performance in large systems.

2.0.1167 `ui_menu_select`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 228d39d076d4308652684c61dd46d71781022466ed70155a37a899acdc69da71

2.0.1168 Function Overview

The `ui_menu_select()` function in Bash is used to create a user interface menu that accepts input from the user and presents a list of selectable options. On selection of a valid option, the function outputs the chosen value and gracefully exits. In case of an invalid selection, it prompts the user for a valid selection until a valid option is selected.

2.0.1169 Technical Description

- **Name:** `ui_menu_select`
- **Description:** A bash function that prints a list of options (menu) to the console, takes user input, and validates the input. If the input is valid, it prints the selected option and returns. If the input is invalid, it asks for a new input from the user.
- **Globals:** None
- **Arguments:**
  - `$1:` The prompt string for the UI menu.
  - `$@:` An array containing the selectable options for the UI menu.
- **Outputs:** Prints to stdout either the prompt and options for the UI menu, the selected option on valid input, or an error message on invalid input.
- **Returns:** Returns 0 on success, no explicit return on failure.

- **Example Usage:**

```
options=("option1" "option2" "option3")
ui_menu_select "Please select an option:" "${options[@]}"
```

2.0.1170 Quality and Security Recommendations

1. Add checks to validate the input arguments—especially check if the supplied options are not empty.
2. Handle signal interrupts for better robustness.
3. It's generally a good practice to avoid the use of `echo -n` since its behavior might be different across different systems.
4. Sanitize error messages to avoid misleading information or potential injection vulnerabilities.
5. Consider adding a timeout for user input to prevent potential denial of service if the script is being run as a server-side script.
6. Use unset to destroy variables that store sensitive data after their use to prevent unintentional exposure or leakage of such data.
7. Exit with an error code on failure instead of just printing an error message to indicate error status to the calling script or function.

2.0.1171 `ui_pause`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 1636563cd29eae7fb011743bbb84e1bd4b67567168166cfc3cd0cf46472383ec

2.0.1172 Function overview

The Bash function `ui_pause()` is designed to introduce a pause in the script execution, requiring a user intervention to continue. This behavior is accomplished by using the `read` command with the `-rp` option, which reads a line from standard input and prompts with a message until input is received.

2.0.1173 Technical description

- **Name:** `ui_pause`
- **Description:** This function uses the `read` command to pause the execution of a script and display a prompt to the user, requesting them to press the [Enter] key to continue.
- **Globals:** None.
- **Arguments:** None.
- **Outputs:** Outputs a message to the user prompting them to press the [Enter] key.
- **Returns:** Does not return a value.
- **Example Usage:**

```
ui_pause
# The script will pause here until the user presses [Enter].
```

2.0.1174 Quality and security recommendations

1. Lack of user input validation: In this function, any key strike will be interpreted as a signal to continue execution. It would be advisable to include some level of user input validation to ensure that only the specific [Enter] key press is given the ability to continue.
2. Error handling: Unexpected errors or exceptions during the function execution are not accounted for, introducing the potential for unexpected behavior and script crashes. A recommended improvement would be to include error handling mechanisms within the function code.
3. Usability: The current function prints a static message which may be unclear to some users or not applicable in all scenarios. Allowing customization of the pause message could improve usability.
4. Return value: Even though this function does not need to return a specific value, for consistency with other Bash functions, it may be beneficial to return a success status (0) after successful execution.
5. Security: As this function does not make use of any user-provided data or values, there are no apparent security risks. However, it's always a good practice to keep security in mind and follow safe coding practices throughout.

2.0.1175 `ui_print_header`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 57e2fede290b133fd0e31c859a63c119ad15a0eea0326adcc61d2c65983dfecc

2.0.1176 Function Overview

The `ui_print_header()` function in Bash is a utility function built for printing headers in terminal-based user interfaces. The function itself is quite straightforward - it accepts a string as an argument which is then displayed as a title within a border of equals signs (=) before and after the title.

2.0.1177 Technical Description

Name: `ui_print_header()`
Description: This function prints a passed title surrounded by a set of equals signs (=) on the lines before and after the title. This creates a clear and visually distinct header within a terminal or console.
Globals: None
Arguments:

- `$1: title` - This is a string placeholder that the function expects. This value is what the function will print out as the header text.
Outputs: This function prints to stdout. The printout consists of an empty line, then a line of equals signs, followed by the title, another line of equals signs, and finally, another empty line.
Returns: Returns nothing.
Example Usage: `ui_print_header "Welcome to My Program"` - Will print:

```
==================================
   Welcome to My Program
==================================
```

2.0.1178 Quality and Security Recommendations

1. Be aware that there are no sanity checks on the supplied argument. The function will print whatever is supplied as an argument, making it susceptible to potentially handling unexpected or rogue inputs. Therefore, consider validating or sanitizing the input on a higher level of function call.
2. This function does not check the length of input strings. If an excessively long string is supplied as an argument it can lead to inconsistent formatting and potentially unreadable headers.
3. The function doesn't use the locale settings to determine the orientation of the symbols. This may cause issues when it is used in locales that use right-to-left writing systems.
4. As the function doesn't return anything it would not be suitable for scenarios where error handling or feedback would be required based on the output of the function.

2.0.1179 `ui_prompt_text`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 698fb0f6a52fc5fb7d346cb2755ab85d4e44cef66d1ac20f8f40870457b2970a

2.0.1180 Function Overview

The `ui_prompt_text` function in Bash is designed for presenting an interactive prompt to users. This function receives two arguments - a prompt message and a default value. The function first displays the prompt, afterwards if the default value is nonempty, it is also displayed in brackets. A colon and a space character are appended to prepare the text for an expected user input. The user's input is read and stored in a variable, `result`. In case of no user input, the default value is returned, otherwise the user's input is returned.

2.0.1181 Technical Description

Feature	Details
Name	ui_prompt_text
Description	A Bash function to prompt users for input with the option of a default response.
Globals	None
Arguments	$1 (prompt): The message prompt to present to the user. $2 (default): The default value that will be used in case of absence of user input.
Outputs	Prompts the user with a message and optional default value.
Returns	The user's input if provided, otherwise the default value.
Example Usage	`ui_prompt_text "Please enter your name" "John Doe"`

2.0.1182 Quality and Security Recommendations

1. Always use `read -r` to prevent interpreting backslashes as escape characters.
2. Beware of potential security risks of command injection if the result is used in further commands without sanitization.
3. You should always quote your variable substitutions like so: `"$var"`. This is to prevent issues with multi-word strings.
4. Remember to initialize local Bash variables. This can help avoid problems if there's a global variable with the same name.
5. Provide clear and user-friendly prompts to facilitate the operation for end users.
6. Default values should be carefully chosen to prevent problems in case of user misuse or misunderstanding.
7. When handling sensitive data, ensure that input is hidden or obscured to protect it from unauthorized access or exposure.

2.0.1183 `ui_prompt_yesno`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 5b56e1af7169ad6721f24f1f595cee61d779f0b22963936d53073e284b177e9c

2.0.1184 Function Overview

The `ui_prompt_yesno` function in Bash is a user interface function that prompts the user with a yes/no question. It loops until the user provides a valid response. The first argument is the prompt text and the second optional argument sets the default answer.

2.0.1185 Technical Description

- **Name:** `ui_prompt_yesno`
- **Description:** This function prompts the user with a question and loops till it gets a valid "yes" or "no" answer from the user. It ensures the users interaction in a script where a binary input is necessary for further execution.
- **Globals:** No global variables are used.
- **Arguments:**
  - `$1`: This is the prompt text that is to be displayed to the user.
  - `$2`: This optional argument specifies the default answer.
- **Outputs:** Outputs the prompt question with an optional default value and user response.
- **Returns:**
  - Returns 0 if the response is "yes".
  - Returns 1 if the response is "no".
- **Example Usage:** If we need user's confirmation for proceeding further, we can use the function as follows:

```
ui_prompt_yesno "Do you want to continue?" "n"
if [ $? == 0 ]
then
    echo "User wants to continue"
else
    echo "User doesn't want to continue"
fi
```

2.0.1186 Quality and Security Recommendations

1. Validate that $1 (the prompt text) exists and is non-empty before proceeding.
2. Regularly audit and update third-party dependencies to keep them up to date with the latest security updates.
3. Avoid using raw user inputs without validation. In this case though, the input is controlled to "y" or "n" only.
4. Use case-insensitive matching to allow inputs as 'Y', 'n', etc.

5. Provide more detailed information about valid inputs for prompt. Do not assume users know they need to enter 'y' or 'n'.
6. Test the function rigorously with a variety of different inputs and edge cases to ensure it handles those correctly.
7. Consider implementing a limit on the number of invalid attempts before automatically selecting the default option.
8. Follow secure code practices and maintain a regular schedule for reviewing/updating the function.

2.0.1187 `unmount_distro_iso`

Contained in `lib/functions.d/iso-functions.sh`

Function signature: e11fdcfe99e864f4cee9a16d324b9d48b2ee3fa3525026c1dd2e8c8f233b0e61

2.0.1188 Function overview

The `unmount_distro_iso()` function has been designed to help unmount a Linux distribution's ISO file. It accepts a distribution identifier or path to the ISO file as an argument and proceeds to determine the mount point and check if this is currently in use. If it's mounted, the function attempts to unmount it, first through a normal unmount command and then via a lazy unmount if the first approach fails.

2.0.1189 Technical description

- **Name**: unmount_distro_iso
- **Description**: Function to unmount a Linux distribution's ISO file.
- **Globals**: None.
- **Arguments**:
  - `$1: os_id_or_path` - A string that represents either the distribution identifier or the absolute path to the ISO file.
- **Outputs**: Logs messages to indicate the unmounting process status (not mounted, successful unmount, lazy unmount, or failed unmount).
- **Returns**:
  - `0`: If the unmount process is successful or the mount point is not in use.
  - `1`: If the unmount process fails.
- **Example Usage**:

```
unmount_distro_iso "/path/to/iso"
unmount_distro_iso "distro_id"
```

2.0.1190 Quality and security recommendations

1. Ensure to validate and sanitize the `os_id_or_path` input to protect against path traversal or any form of injection attacks.

2. Always check and handle error conditions appropriately. In the situation where both normal and lazy unmount fails, the function should handle this condition properly and not proceed assuming the unmount was successful.
3. Consider adding more descriptive logging messages to help with debugging if an error occurs.
4. Validate the provided mount point before trying to unmount. Confirm that it exists and is indeed a mount point to prevent unintended behavior.
5. Implement some form of logging system that handles log rotation and log levels. This would prevent your disk space from being filled up by logs and control the verbosity of your logs.

2.0.1191 `update_distro_iso`

Contained in `lib/functions.d/iso-functions.sh`

Function signature: a0dbc05086e473d99eb0bcb51d5eebbb8b04237c7f7d828c90a17cc8eceac8f1

2.0.1192 Function overview

The bash function `update_distro_iso()` is used to update the ISO file of the provided Linux distribution string. The function first unmounts the current ISO, prompts the user to update the ISO file, and then re-mounts it. The function takes one argument, `DISTRO_STRING`, and uses it to construct the ISO file and mount point location paths. If the function encounters any issues, such as if the provided distribution string is empty, the mount point remaining after attempting to unmount, or the ISO file does not exist, it returns an error to the command line and exits with a status code of 1.

2.0.1193 Technical description

- **Name:** `update_distro_iso()`
- **Description:** This shell function unmounts a specified Linux distribution ISO, prompts the user to update the ISO, and then remounts it.
- **Globals:** No globals are used in this function.
- **Arguments:**
  - `$1: DISTRO_STRING` Description: A string that identifies the Linux distribution ISO (format: `<CPU>-<MFR>-<OSNAME>-<OSVER>`).
- **Outputs:** Informational and error messages directed to the command line.
- **Returns:** `1` if any errors occur, such as the DISTRO_STRING argument missing, the mount point still being present after attempting to unmount, or the ISO file not existing.
- **Example Usage:** `update_distro_iso "x86-Intel-Ubuntu-20.04"`

2.0.1194 Quality and security recommendations

1. This function assumes specific directory structure and filenames based on the `DISTRO_STRING`. It would be more robust and secure if it independently verified the path and filename before proceeding.
2. To improve function's robustness, handle other potential errors, such as permission issues when attempting to unmount or mount the ISO file, or non-standard user input.
3. Function currently has no mechanism for verifying that the updated ISO file is valid or correctly formatted. Incorporate a verification step after the user updates the ISO.
4. To further enhance security, sanitize the user input to prevent command injection or make sure that the user input does not contain any special shell characters.

2.0.1195 `update_dns_dhcp_files`

Contained in `lib/functions.d/dns-dhcp-functions.sh`

Function signature: 8f9c1f6b531e8690e223f4019ebfe67747cb7bd35d0aed34a6efcd8fdcbe6b23

2.0.1196 Function Overview

The function `update_dns_dhcp_files()` main job is to update the DNS and DHCP configuration files. It uses two other functions `build_dhcp_addresses_file` and `init_dns_hosts_file` to do so. If either of these functions fail, an error is logged and the function returns 1. If they both succeed, the local resolv.conf file is updated using the `_ips_resolv_conf_update` function and an informational message is logged indicating the successful completion of the function.

2.0.1197 Technical Description

- **Name:** update_dns_dhcp_files
- **Description:** This function is used to update the DNS and DHCP configuration files. It performs the update task by invoking build_dhcp_addresses_file and init_dns_hosts_file functions. If any of these function calls do not succeed, it logs the error and returns 1 indicating the failure. On successful execution of the aforementioned function calls, another function _ips_resolv_conf_update is invoked to update the local resolv.conf file, after which it logs the successful completion of its execution.
- **Globals:** None
- **Arguments:** None
- **Outputs:** Logs informational and error messages.
- **Returns:** 0 on success, 1 on failure.
- **Example Usage:** `update_dns_dhcp_files`

2.0.1198 Quality and Security Recommendations

1. To enhance error tracking, consider adding more descriptive error messages in the logging function, `hps_log`. This could assist users to debug issues better.
2. Try to minimize the use of globals. If the helper functions `build_dhcp_addresses_file`, `init_dns_hosts_file` and `_ips_resolv_conf_update` rely on global variables, consider alternatives such as passing parameters or return values.
3. For a more robust design, you could consider double checking if the DNS and DHCP configuration files are updated correctly after the function executed successfully.
4. Document the helper functions `build_dhcp_addresses_file`, `init_dns_hosts_file` and `_ips_resolv_conf_update`.

2.0.1199 `url_decode`

Contained in `lib/functions-core-lib.sh`

Function signature: 0c9d59a1abf00641484dc3cdac7c213954f15d503815ab354b463a64b7d808a6

2.0.1200 Function overview

This function, `url_decode()` mainly decodes the URL provided as input. It initialises the `data` variable by replacing '+' in the input with spaces. Then it prints the `data` replacing the '%' signs with ascii codes. It also gets the origin identifier from the hostname if available or else uses the origin tag. It then decodes the message, checks if rsyslog is running and if yes, it logs the message directly or writes to a file if logging is not possible.

2.0.1201 Technical description

- **Name**: `url_decode()`
- **Description**: This function is used to decode a URL. It gets the origin identifier, decodes the message, checks for rsyslog, and logs the message either through logger if rsyslog is running or by writing directly to the file. If writing is not possible, it logs to stderr.
- **Globals**: `origin_tag:(hps_origin_tag())`, `origin_id:(the origin identifier either from hostname or origin tag)`, `msg:(the decoded message)`, `rsyslog_running:(checks if rsyslog is running)`.
- **Arguments**: `$1:(the URL to be decoded)`, `$2:(the raw message to be decoded)`.
- **Outputs**: Logs the decoded message.
- **Returns**: `0`.
- **Example usage**: `url_decode "http%3A%2F%2Fexample.com%2F"`

2.0.1202 Quality and security recommendations

1. Check the validity and correctness of the input URL before decoding.
2. Make sure to properly handle special characters in the URL to prevent any issues during decoding.
3. Consider adding error checking for each step like retrieving origin tag, decoding message, checking rsyslog status, logging message etc.
4. Ensure message logging and output writing processes are secure and the initiated processes cannot be hijacked.
5. Always sanitize and check your inputs and outputs, never trust user-generated inputs.
6. Always update the paths and software used in checking to ensure they are up to date for the latest security patches.

2.0.1203 `urlencode`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: f758d39e7a343eef82fc4e92ae1358118cf9a79d8accf9f5013313b5448282ac

2.0.1204 Function Overview

The `urlencode` function is a utility function used to encode a string by converting certain characters into their hexadecimal representation prefixed by `%`. The function operates by iterating over each character in the source string and checking if it falls within certain ranges. If it doesn't, that character is replaced by its encoded form.

2.0.1205 Technical Description

- **Name:** `urlencode`
- **Description:** The `urlencode` function is designed to encode a string by replacing certain characters with their URL-encoded form based on the ASCII character set. For every character not in the set `[a-zA-Z0-9.~_-]`, it is replaced with its hexadecimal ASCII value prefixed by `%`.
- **Globals:** None
- **Arguments:**
  - `$1`: This is the string that needs to be URL-encoded.
- **Outputs:** The function prints the URL-encoded version of the input string.
- **Returns:** None
- **Example Usage:**

```
$ urlencode "Hello, World!"
Hello%2C%20World%21
```

2.0.1206 Quality and Security Recommendations

1. Validate the inputs: Before processing, validate that the input is in fact a string and not any other data type to avoid errors during and unexpected results from processing.
2. Error handling: The function currently lacks error handling. Add an error message or an error code to handle situations where an invalid input is given.
3. Unit testing: Ensure each piece of this function is adequately tested, both with expected and unexpected inputs to ensure it behaves as expected in all scenarios.
4. Use of `local`: This function appropriately uses `local` variables to ensure that they do not clash with variables outside of the function. This should be continued for any new variables introduced into the function.
5. Security: The function as is does not have any direct security concerns. However, always be aware of potential security risks when dealing with URL encoding, especially in web development contexts where URL encoded strings can sometimes be manipulated by malicious actors.

2.0.1207 `validate_alpine_repository`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: 6896b5eb251bde169fdbd07da3c4c963f3fae01dd749ae9de9c1298ffd83e4eb

2.0.1208 Function Overview

The function `validate_alpine_repository()` is used to validate an Alpine Linux package repository. It requires an Operating System (OS) ID and a repository name as inputs and ensures that the repository directory and the APKINDEX file exists. It also checks the number of available packages in the repository, and compares this count with the expected minimum count. If any of these verifications fail, it logs an error and returns 1. If all verifications pass, it logs the successful validation and returns 0.

2.0.1209 Technical Description

The `validate_alpine_repository()` can be described as follows:

- **name**: `validate_alpine_repository`
- **description**: Validating an Alpine Linux repository based on OS ID and repository name.
- **globals**: [`hps_log`: Function used for logging]
- **arguments**: [`$1`: OS ID, `$2`: repository name (defaults to 'main')]
- **outputs**: Logs the current processing status and any errors encountered in the repository validation.
- **returns**: `0` if the repository validation is successful, or `1` if it fails.
- **example usage**: `validate_alpine_repository "alpine" "main"`

2.0.1210 Quality and Security Recommendations

1. Enable error handling modes (`set -o errexit`, `set -o nounset`, `set -o pipefail`) to handle potential errors and unforeseen conditions.
2. Use functions for complex operations to improve code readability and maintainability.
3. Validate user-provided inputs to prevent potential vulnerabilities and unexpected behavior.
4. Centralize all error logging to ensure a consistent and informative error reporting system.
5. Regularly update and audit your codebase for potential security flaws and efficiency improvements.

2.0.1211 `validate_apkindex`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: 8b01a88e6110ddb2342de0b3ad1c7ac2c81b45e6b739b8d49cb390eeb12b97d9

2.0.1212 Function Overview

The `validate_apkindex` function is designed to check an APKINDEX.tar.gz file in a given directory. This file is a compressed tag-separated file used by Alpine Linux package management to store metadata about packages in an Alpine repository. It first checks whether such a file exists within the given directory, and then checks if said file is corrupt or not. If the file does not exist or has been corrupted, it will return an error. Once the file has been validated successfully, the function returns 0.

2.0.1213 Technical Description

```
- **Name**: `validate_apkindex`
- **Description**: Validates the availability and integrity of the APKINDEX.tar.gz fil
- **Globals**: None
- **Arguments**:
   - `$1`: `repo_dir` - The directory where the APKINDEX.tar.gz file is expected to be.
- **Outputs**: Logs either a successful validation of APKINDEX.tar.gz, or logs errors
- **Returns**:
   - `2` if the APKINDEX.tar.gz file is not found or is corrupted.
   - `0` if the APKINDEX.tar.gz file is successfully validated.
- **Example usage**:

```bash
validate_apkindex "/path/to/directory"
```
```

```
### Quality and Security Recommendations

1. Provide clear and explicit error messages that can be acted upon without revealing s
2. Consider adding file permissions checks to ensure that the file can be accessed by th
3. Where possible, avoid using global variables to avoid potential conflicts and increa
4. Explicitly declare input expectations to help prevent potential manipulation and mi
5. Always exit with a non-zero status code when a failure occurs to allow other scripts
6. Regularly check for and manually handle potential errors and exceptions in your scri

### `validate_hostname`

Contained in `lib/functions.d/network-functions.sh`

Function signature: 1b48bafa4d6a9144287a36e810a96b33fe5fc9b78c467e672aa1e7c0186f54b

### Function overview

This bash function, `validate_hostname`, undertakes the task of validating a hostname

### Technical description

**Function details:**

- **Name**: `validate_hostname`
- **Description**: Verifies if the given hostname complies with the permissible condit
- **Globals**: None
- **Arguments**:
  - `$1: hostname` - Hostname to be validated.

- **Outputs**: No explicit output; all outputs are implied through return codes.
- **Returns**:
  - `0` - If the hostname complies with all conditions.
  - `1` - If the hostname does not comply with any condition.

- **Example usage**: `validate_hostname google.com`

### Quality and Security Recommendations

1. Consider using explicit error messages to elaborate on the reason for validation fai
2. Use principles of least privilege for any direct access to system level resources, s
3. Keep an eye on performance and compliances with large input values.
4. Implement unit tests to ensure the functionality of this function is as expected.
5. Include a logging mechanism to audit the program's activity which helps in troublesh
```

```
### `validate_ip_address`

Contained in `lib/functions.d/network-functions.sh`

Function signature: e46beef7993583a0fd0da40d09a03ba29d295b6eea7552ec53c3a8434a1b7eb

### Function Overview

The function `validate_ip_address` is a bash function that is used for validating that

### Technical Description

- **name:** validate_ip_address
- **description:** Validates that a string is a valid IP address.
- **globals:** [ No global variables used ]
- **arguments:** [ $1: String to validate as IP address ]
- **outputs:** None
- **returns:** Returns 1 if the IP is not valid, and 0 if the IP is valid.
- **example usage:**

```bash
validate_ip_address "127.0.0.1"  # Valid; returns 0
validate_ip_address "999.0.0.1"  # Not valid; returns 1
```

2.0.1214 Quality and Security Recommendations

1. **Input Validation:** The function should handle edge cases where the input is either empty or an invalid data type. This prevents unexpected behavior and potential script vulnerabilities.
2. **Error Messaging:** Instead of simply returning 0 or 1, consider including an informative error message if the provided IP address is invalid. This would make the script more user-friendly and easier to troubleshoot.
3. **Documentation:** Each section of the code should be well commented for easier maintenance and readability. This includes the function's purpose, its input and output, and how it handles different conditions.
4. **Unit Tests:** Create unit tests to cover different edge cases and function behaviors. This will ensure the function works as expected and helps identify bugs.

2.0.1215 `verify_checksum_signature`

Contained in `lib/functions.d/iso-functions.sh`

Function signature: 56bea15c9139a6a7e8a5f6c061d1ea0f9983469c94f7d3149e7ccc3bf4984e51
```

2.0.1216 Function Overview

The function `verify_checksum_signature()` is used to verify the integrity of an ISO file using checksum validation and GPG signature checking. It first asserts the existence of the ISO file, failing if the file does not exist. It then fetches CHECKSUM and `CHECKSUM.sig` files for the respective OS from a defined URL. Afterwards, it imports the necessary GPG key and verifies the GPG signature on the fetched CHECKSUM, and the actual checksum of the ISO file against the fetched CHECKSUM. The function has a case for `rockylinux` and fails by default for other OS.

2.0.1217 Technical Description

- **Name**: `verify_checksum_signature`
- **Description**: This function verifies the checksum and GPG signature of ISO files.
- **Globals**:
  - No globals used by this function
- **Arguments**:
  - `$1`: cpu - architecture of the targeted machine
  - `$2`: mfr - manufacturer of the targeted machine
  - `$3`: osname - name of the OS represented in the ISO file
  - `$4`: osver - version of the OS represented in the ISO file.
- **Outputs**:
  - Errors to stderr when required files are not found, when downloads fail, when an invalid GPG key is imported, when the GPG signature verification fails, when the checksum mismatched, or when checksum verification is not implemented for the provided OS.
  - Verification status to stdout.
- **Returns**: `1` when a failure situation is encountered, `0` when the ISO file passes verification.
- **Example usage**: `verify_checksum_signature amd64 Lenovo rockylinux 8`

2.0.1218 Quality and Security Recommendations

1. To improve security, validate the URLs before using them in `curl`. It is common for URLs to contain injection points that can lead to shell command injection if not sanitized properly.
2. Maintain a good error handling system, don't just rely on the return status. Log errors for troubleshooting.
3. Implement checks for more Linux distributions, rather than just for `rockylinux`.
4. Create a cleanup routine to remove any temporary files and directories created even if the function fails or is interrupted.
5. Use HTTPS for all URLs to ensure secure transfer of files.

6. Use the long option names (e.g., `--silent` instead of `-s`) with the curl command for better readability.

2.0.1219 `verify_required_repo_packages`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: a16392408f8b201975834d3d38029e3859302d45f825b35156dca0ae85e71609

2.0.1220 Function Overview

The `verify_required_repo_packages` function checks for the presence of certain required packages in a specified repository path. It is intended for use with RPM-based package repositories. The function takes the repo path as the first argument, then a list of required package names. If the function cannot find a required package in the repository, it logs an error message and returns a value of 1 or 2, depending on the type of error. If all required packages are present, it logs an informational message and returns a zero value.

2.0.1221 Technical Description

- **Name**: `verify_required_repo_packages`
- **Description**: This function inspects a specified package repository and verifies the presence of required packages. It is used for checking the availability of certain essential packages in a RPM repository.
- **Globals**: None.
- **Arguments**:
  - `$1: repo_path` - The path to the package repository.
  - `$2...: required_packages` - An array of package names that the function will check for in the repository.
- **Outputs**: Logs error messages through `hps_log` function if required packages are missing or repository path is not provided. Logs an informational message if all required packages are present.
- **Returns**:
  - `0` if all required packages are present in the specified repo_path.
  - `1` if repo_path not provided or does not exist.
  - `2` if any of the required packages are missing.
- **Example Usage**: `verify_required_repo_packages "${HPS_PACKAGES_DIR}/${DIST_STRING}/R zfs opensvc`

2.0.1222 Quality and Security Recommendations

1. It is recommended to use absolute paths for `repo_path` to avoid any ambiguity or errors derived from relative paths usage.

2. Ensure the proper access rights are in place for the directory path specified by `repo_path`, so the function can process the commands effectively.
3. Always sanitize input given to the function to prevent potential security vulnerabilities, such as command injection.
4. Consider adding further error checking mechanisms, like checking if each package name in `required_packages` is a non-empty, non-null string.
5. Implement a feature to handle version-specific packages in the array `required_packages`. Currently, it assumes that the requirement is met if any version of the package exists.
6. Leverage the return status of the function to handle error situations in the script that calls this function.

2.0.1223 `verify_rocky_checksum_signature`

Contained in `lib/functions.d/iso-functions.sh`

Function signature: c29ccb47fbd2f30ffd999a0dbdad086aae4efbf33487b193102452f5f4b62cab

2.0.1224 Function Overview

The function `verify_rocky_checksum_signature()` is designed for verifying the checksum and signature of the downloaded Rocky Linux ISO in order to ensure its accuracy and safety. It requires a version number as an argument to specify which version of Rocky Linux for validation. The function firstly downloads the GPG key from Rocky Linux's official website. Then it uses this GPG key to verify the signature of the downloaded ISO CHECKSUM. Finally, it calculates and compares the SHA256 checksum of the downloaded ISO with the expected checksum obtained from the server.

2.0.1225 Technical Description

- **Name:** `verify_rocky_checksum_signature()`
- **Description:** This function verifies the GPG signature of the CHECKSUM file of a particular downloaded Rocky Linux ISO. It also computes the SHA256 checksum of the downloaded ISO file and compares this calculated checksum against the server provided expected checksum.
- **Globals:**
  - `arch: x86_64` Architecture of the system
  - `base_url: https://download.rockylinux.org/pub/rocky/${version}/${arch}/iso/` The base url from which the ISO and other resources will be downloaded
  - `target_dir: "$(_get_distro_dir)/rocky"` The directory where the ISO is stored after downloading
- **Arguments:**
  - `$1: version` The version of Rocky Linux ISO to be verified

- **Outputs:** Status messages about ongoing process, checksum mismatch errors, GPG key import errors and GPG signature verification errors
- **Returns:** 0 when GPG signature and checksum verification are successful, 1 when GPG key import fails, 2 when GPG signature verification fails, 3 when expected checksum is not found in the downloaded CHECKSUM file, 4 when the received and computed checksums of the ISO mismatch.
- **Example Usage:** `verify_rocky_checksum_signature 8.4`

2.0.1226 Quality and Security Recommendations

1. Implement additional error handling and logging to provide more detailed feedback if the GPG key download fails or if incorrect input is provided.
2. Before proceeding, validate the URL where the GPG key will be downloaded.
3. Introduce tests to ensure SHA256SUM algorithm's reliability used in the computation of the received ISO checksum.
4. Update the GPG key in a regular basis in case it has been updated by the server and consider a secure key handling.
5. Make sure the directory and the ISO file path are valid and secured and have the write permission.
6. Check the availability of the `curl`, `gpg` and `sha256sum` tools before execution.

2.0.1227 `version_compare`

Contained in `lib/functions.d/os-function-helpers.sh`

Function signature: 1cf12ac889d82a73e0a55fdba068893c38485656aaf7dced684dd0582fdfb215

2.0.1228 Function overview

The `version_compare` function is designed to compare two version strings. It takes into account different versioning naming schemes by parsing and padding the version string to make them comparable. This is especially useful for software versioning or system builds where it is important to know the relationships between different versions, such as if one is greater than, less than, or equal to another.

2.0.1229 Technical description

- **Name:** `version_compare`
- **Description:** Compares two version strings in a function. It also provides options for different comparison operators.
- **Globals:** None
- **Arguments:**
  - `$1`: This is the first version number that needs to be compared.

  - `$2`: This is the comparison operator being used for the comparison. All reasonable cases are accounted for, such as greater than (>), less than (<), and equals (=).
  - `$3`: This is the second version number that needs to be compared with the first one.

- **Outputs:** The result of the comparison is returned.
- **Returns:**
  - `0` if the comparison matches with the operator.
  - `1` in all other cases.
- **Example usage:**

```
version_compare "1.10.1" "<" "1.10.2"
# Will return 0 (true), as first version number is less than the second one.
```

2.0.1230 Quality and security recommendations

1. Be sure to validate and sanitize all inputs. This function does not check for invalid characters or string formatting in version inputs.
2. Add more extensive error handling, for example, in the cases where a non-version string is entered.
3. Secure the function against potential DoS attacks by limiting the size of the version string that can be passed to the function.
4. Create unit tests to ensure the function operates as expected in all circumstances and edge cases.
5. Always use the latest version of Bash to ensure the highest level of security and performance.

2.0.1231 `write_cluster_config`

Contained in `lib/functions.d/cluster-functions.sh`

Function signature: cf6f48116ee4f6ae2fa075ed74a4476f6c22ffe594564ebf704d40d3dce09039

2.0.1232 Function overview

The `write_cluster_config` function is designed to write a series of values (array) into a targeted configuration file. The function starts by checking the length of the array of values, and if empty, outputs an error message and returns `1` (indicating an error occurred). If the array is not empty, the function prints the values to the terminal and then writes the inter-space-separated values into the targeted file.

2.0.1233 Technical description

- **Name:** write_cluster_config

- **Description:** Writes an array of values to a targeted configuration file. Reports an error and returns 1 if the array is empty. Otherwise, prints the array of values to the screen and writes them to the file.
- **Globals:** None
- **Arguments:** [$1: Target file for writing the array, $2: The array of values]
- **Outputs:** "[x] Cannot write empty cluster config to $target_file" to stderr if the array is empty; otherwise, "Writing: ${values[*]}" and "[OK] Cluster configuration written to $target_file" to stdout.
- **Returns:** Returns 1 if the array is empty. Does not explicitly return a value otherwise.
- **Example usage:**

```
write_cluster_config "config.txt" "value1" "value2" "value3"
```

2.0.1234 Quality and security recommendations

1. Consider validating file write operations: While the function currently reports whether a configuration file is written, it could potentially add error checking for the file write operation to catch and report errors.
2. Input validation: More robust validation of input arguments (such as checking if $1 is a valid file path) will help prevent accidental misuse of the function.
3. Atomic writes: Consider using atomic write operations to prevent potential race conditions or half-written files in case of errors or interruptions during write operations.
4. Secure handling of error messages: Rather than writing error messages to stderr, consider logging them securely in a way that would not expose potentially sensitive information.
5. Sanitization of inputs: Always sanitize inputs especially if they are used as part of a command to be executed to avoid command injection vulnerabilities.

3 Node functional reference

Functions available only on nodes

3.0.1 `build`

Contained in `node-manager/alpine-3/TCH/BUILD/10-build_opensvc.sh`

Function signature: 56e136787268e5be7a6960ee47b56f75a2c85fab6db9a387f30a81584efcfb76

3.0.2 1. Function Overview

The function `build()` is currently a placeholder with no processing logic. It strictly returns a static value `0`, indicating successful execution.

3.0.3 2. Technical Description

3.0.3.1 Name

`build`

3.0.3.2 Description

An utility function designed to be expanded with future logic. At present, it does not take any arguments or act on any global variables. When invoked, it simply returns a `0` status code - generally interpreted in Unix-like environments as indicating the successful completion of a task.

3.0.3.3 Globals

None. The function does not reference or modify any global variables.

3.0.3.4 Arguments

None. The function does not take any arguments.

3.0.3.5 Outputs

None. The function does not output anything.

3.0.3.6 Returns

`0`. This is a standard signal of successful execution in a Unix-like environment.

3.0.3.7 Example Usage

```
status=$(build)
echo $status
```

3.0.4 3. Quality and Security Recommendations

1. **Specify Function Purpose**: Right now, the function does not do anything except report a success status. When it is expanded, records should be kept of its design purpose and expected input/output patterns.
2. **Check for Argument Existence**: If arguments are added, these should be checked to ensure they exist within the function. This can be done using conditional guards.
3. **Validate Inputs**: Depending on the nature of the arguments added, it may be appropriate to validate inputs against a range of acceptable values.
4. **Handle Errors**: If there is any potential for error (such as a bad input or a failed subprocess), these errors should be elegantly caught and handled.
5. **Secure Return Values**: Ensure that the function always return values so function callers can handle responses appropriately. Currently, the function only returns `0` which limits error handling capabilities.
6. **Encrypt Sensitive Data**: If the function begins working with sensitive data, storage and transmission must be secured.
7. **Debug Information**: Add print or echo statements during the debugging and testing phase to ensure the function operates as expected. These should be removed or toggled via a verbose mode once the function is live.

3.0.5 `build_zfs_source`

Contained in `node-manager/rocky-10/rocky.sh`

Function signature: b965b327fc7d07973555902b7baf3453c683f2b38bed7e1b9b7459d86b9bb261

3.0.6 1. Function overview

The `build_zfs_source` function facilitates the downloading and building of ZFS from a source, using the DKMS method. The function commences by requesting for an index file from a server, from which it determines the exact ZFS source file to download. Following a successful download, the function uncompresses the source file and installs required build dependencies. It then builds and installs the ZFS module, verifying if the module exists in the system right after installation.

3.0.7 2. Technical description

- **Name:** `build_zfs_source`
- **Description:** Downloads and builds the ZFS source code from the server.
- **Globals:** No global variables are directly manipulated by this function.
- **Arguments:** This function doesn't take any arguments.
- **Outputs:** Variable log outputs are presented to the standard output, depicting the progression of download, extraction, building, and installation processes.
- **Returns:**
  - `0` if the ZFS build and installation is successful.
  - `1` if any of the interim steps fail (download index, fetch source file, install dependencies, extract archive, configure, build, install, or verify ZFS module).
- **Example usage:** To use this function, it can simply be called without any arguments as follows: `build_zfs_source`

3.0.8 3. Quality and security recommendations

1. Consider improving error handling by catching and handling exceptions at more granular levels.
2. Make use of secure methods to download files from the server. The links to the files should be encrypted with HTTPS to ensure secure data transmission.
3. The function could benefit from more thorough input validation (not applicable in this scenario, but highly recommended in cases where user input is involved).
4. The process of building and installing ZFS manually from source is complex and might pose a security risk if not handled properly. Consider using packaged versions of ZFS where security fixes and package maintenance are managed by the distribution.
5. Temporary files are created at `/tmp`. Instead, a hardcoded directory should be avoided, instead leverage built-in features for making temporary files/directories.
6. Implement a log function that records all the activities that occurred during the process and any error messages thrown.

3.0.9 `check_zfs_loaded`

Contained in `node-manager/rocky-10/zpool-management.sh`

Function signature: e83726e842477cf79c67cdcf1de046b56eacfafd7f97ba195d70b1f2bccfabc2

3.0.10 Function overview

The function `check_zfs_loaded` is responsible for detecting the presence of the ZFS command and its kernel module in the system. It first verifies if the `zfs` command is available. If it is not found, the function ends with a status code `1`. If the command is available, it is checked if the ZFS module is currently loaded in the system. If the module

is not loaded, an attempt is made to load it using `modprobe`. If the loading operation fails, the function returns 1, signaling the problem to the caller.

3.0.11 Technical description

- **name**: `check_zfs_loaded`
- **description**: Checks whether the `zfs` command is available and if the ZFS kernel module is loaded. If the ZFS module is not loaded, tries to load it.
- **globals**: None
- **arguments**: None
- **outputs**: Diagnostic messages on console about the availability of `zfs` command and the ZFS kernel module status.
- **returns**: 0 if `zfs` command is available and ZFS kernel module is loaded or was able to load; 1 otherwise.
- **example usage**: `check_zfs_loaded`

3.0.12 Quality and security recommendations

1. Avoid using `echo` to pass error messages. Instead, use the STDERR stream to prevent interfering with STDOUT.
2. Ensure the script is run by a user with sufficient permissions, especially for executing commands like `modprobe`. Validate the user's permissions before running such commands.
3. Apply error handling to catch and handle unexpected issues properly. Consider the potential failures of each command used in the function.
4. Document the function and its expectations more comprehensively. The script's users or maintainers will benefit from a clear description of its expected inputs, outputs and error cases.
5. Potentially dangerous operations, like loading a kernel module, should be performed with caution. Confirm this action with the user before proceeding.

3.0.13 depend

Contained in `node-manager/alpine-3/TCH/BUILD/10-build_opensvc.sh`

Function signature: 89406247984185d00547bde8e948365eab651f57a6d007cee8af08dc83a26713

3.0.14 1. Function Overview

The `depend` function is a utility function used within a script to set software or service dependencies. This function is responsible for specifying the dependencies that are needed in various parts of the shell script. These dependencies can be either software libraries, system mechanisms, or any other component that the script might depend

on. depend accomplishes this by making use of four other function calls: need, use, after, and blk-availability.

3.0.15 2. Technical Description

- **Name**: depend
- **Description**: The function is used to specify dependencies that are needed for different parts of a shell script to work well. It uses four other function calls namely, need, use, after, and blk-availability.
- **Globals**: N/A
- **Arguments**: The function does not directly act on arguments. Instead, it includes these other functions that use arguments to specify particular dependencies.
  - need: This function takes a single argument, net, which means that network services are needed.
  - use: This function requires several services namely, docker, libvirtd, libvirt-guests, blk-availability, and drbd as arguments.
  - after: This function requires the time-sync service as an argument.
- **Outputs**: There are no explicit outputs. The function will trigger error messages if the dependencies are not present.
- **Returns**: There are no explicit return values. Successful completion of the function means that all dependencies are in place.
- **Example usage**:

  ```
  depend
  ```

3.0.16 3. Quality and Security Recommendations

1. Ensure all dependencies specified in the function calls are indeed required for the script in question.
2. Keep the dependencies up-to-date, security vulnerabilities often occur in outdated dependencies.
3. The dependencies should be tested for potential conflicts. Since the dependencies are being used together in the same setting, they must be compatible.
4. It is good practice to document the role of each dependency within the larger script to enable easy troubleshooting and maintenance.
5. Ensure proper error handling is in place in case a dependency is missing or fails to load.
6. Keep the function and scripts clear of any sensitive information like keys or passwords. This can prevent potential security breaches.

3.0.17 disks_free_list_simple

Contained in node-manager/rocky-10/zpool-management.sh

Function signature: 23bd3ac7877a7c90a934c9f11fb7a056bf39f675410585baaaa84e107157944c

3.0.18 1. Function Overview

The function disks_free_list_simple is a Bash script function that aims to enumerate disks that are not in use on a Unix-based system. These disks are available for storing data. If a disk meets certain criteria, it is treated as free. The function uses various Unix commands like lsblk, awk, grep, readlink and zpool to process and filter the disk information. It aims to disregard certain types of disks, like those which are either removable, part of a raid setup, lun storage, loop type or mounted.

3.0.19 2. Technical Description

- Definition:
  - **Name**: disks_free_list_simple
  - **Description**: A Bash function that lists the unused disks in a Unix/Linux based system.
  - **Globals**: None
  - **Arguments**: None
  - **Outputs**: Prints the list of unused disks to standard output (stdout).
  - **Returns**: No explicit return value, uses the default return of the last statement executed.
  - **Example Usage**: Call the function without parameters like so: disks_free_list_simple

3.0.20 3. Quality and Security recommendations

The following are suggested quality and security improvements for the function:

1. Using more descriptive variable names improving readability and context to future developers.
2. Add data validation and error checking wherever possible.
3. Commenting or documenting the function and its logic for better understandability.
4. Avoiding the use of readlink without the -e or -f option to prevent potential mishandling of non-existent file or directory inputs.
5. Adding more specific filters and criteria for the disks to be included in the unused list.
6. For security, consider handling permissions that allow script execution and managing sensitive data.

7. Optimizing the function by reducing command line calls or pipe operations. Use internal shell operations where possible.
8. Standardizing the way inputs and outputs are handled extending the function's versatility in various circumstances.

3.0.21 `_extract_apk_version`

Contained in `node-manager/alpine-3/alpine-lib-functions.sh`

Function signature: eb0d9b3cde2db7f28ef70b36d8d6c4d6407f51ef1ca999daff92c5de88823f30

3.0.22 Function Overview

The function `_extract_apk_version` is used to extract an Application Package (APK) version from a given filename. The function accepts two arguments, the filename and the package name, and processes the filename to remove the package name prefix and the .apk suffix, thus returning the version of the APK.

3.0.23 Technical Description

- **Name**: `_extract_apk_version`
- **Description**: Extracts the version information from an APK filename by removing the package name prefix and the .apk suffix.
- **Globals**: None.
- **Arguments**: [$1: `filename` (name of the APK file), $2: `pkg_name` (name of the package)]
- **Outputs**: Version information extracted from the filename.
- **Returns**: Returns the version information string.
- **Example usage**:

```
_extract_apk_version package-1.0.0.apk package
# output: 1.0.0
```

3.0.24 Quality and Security Recommendations

1. Consider validating the inputs: Currently, the function doesn't check if the inputs are valid before processing them. You should validate whether the filename and package name provided follow the expected format.
2. Adding error handling: The function doesn't have any error handling if the filename or package name doesn't match the expected structure. It'd be beneficial to add some error handling to let the user know why the function might not be working as expected.
3. Sanitization of inputs: Although Bash scripting does not pose traditional vulnerabilities to injection attacks as seen in other forms of programming, it is still good practice to sanitize inputs to avoid unexpected behaviors.

4. Usage of local variables: The function already uses local variables which is good for encapsulation but its usage should be continued to avoid inadvertent changes in global variables.
5. Write a more descriptive comment: The current comment only briefly describes the function's purpose. A better comment might also mention the format of the filename, what inputs are expected, and what output is given.

3.0.25 `_find_latest_apk`

Contained in `node-manager/alpine-3/alpine-lib-functions.sh`

Function signature: 1def432fd9e65b420473bc4af6cfac04edca42e312a4647d7f673bb7dcd41492

3.0.26 Function overview

The `_find_latest_apk` function is designed to find and download the latest version of a specified APK (Alpine Linux package) from a list of available packages. The function takes in the name of an APK package and returns the name of the latest version of the package. If no package is found, the function will return 1, signifying an error.

3.0.27 Technical description

3.0.27.1 Name

`_find_latest_apk`

3.0.27.2 Description

The function `_find_latest_apk` helps in finding the latest version of a specified APK package from a defined list of available packages. It takes into account the naming convention of APK versions (version-rN), and uses Alpine's version comparison logic to find the latest version.

3.0.27.3 Globals

- `available_packages`: A string containing the names of available APK packages.

3.0.27.4 Arguments

- `$1`: Name of APK package to search for its latest version.

3.0.27.5 Outputs

- Echoes the name of the latest version APK package.

3.0.27.6 Returns

- `1` if no matching packages were found.
- `0` if the function executes successfully.

3.0.27.7 Example usage

```
_find_latest_apk bash
```

3.0.28 Quality and security recommendations

1. Validate the package name. The function should validate that the input package name contains only permissable characters, which would mitigate potential code injection attacks via the package name.
2. Error Handling. The function might be more robust if more advanced error handling was implemented, for instance the function could also handle undecipherable package versions or packages with malformed names.
3. Source Traceability. The sources from which the APK package versions are retrieved should be trusted and verified to ensure only genuine packages are considered. This could help prevent security issues caused by fake or malicious packages.
4. Assurance of Access Controls. Ensure the account running the script has the minimum required permissions, this is a well-regarded best practice for improving security.
5. Security of Temporary Files. Temporary files created as part of the function should be securely handled and properly deleted after their usage.

3.0.29 `get_dst_dir`

Contained in `node-manager/alpine-3/TCH/BUILD/run_osvc_build.sh`

Function signature: a90d8dd365ef92d8a352f764798d19209b4fc64ef2e59c431bd6f7a5870c005f

3.0.30 Function Overview

The bash function `get_dst_dir` defined below is a simple function with no arguments, that when called, prints the string "/srv/build/opensvc-om3" to the standard output.

```
get_dst_dir () {
  echo "/srv/build/opensvc-om3"
}
```

3.0.31 Technical Description

- **Name:** `get_dst_dir`

- **Description:** This function prints a hard-coded string "/srv/build/opensvc-om3" to the standard output. It could be used to provide a consistent directory path across multiple scripts.
- **Globals:** None.
- **Arguments:** None.
- **Outputs:** If successful, this function prints the string "/srv/build/opensvc-om3" to the standard output.
- **Returns:** None.
- **Example usage:**

```
destination=$(get_dst_dir)
echo $destination
```

3.0.32 Quality and Security Recommendations

1. **Parameterize the Function:** Instead of hard-coding the string "/srv/build/opensvc-om3", consider passing it as an argument to the function. This would make the function more flexible and reusable.
2. **Error Handling:** Add error handling or checks to the function. For example, check if the directory exists before trying to use it.
3. **Return codes:** Even though this is a simple function, it's a good practice to return a status code. This could help in case the function e.g., fails to print the output due to a memory issue.
4. **Documentation:** Ensure that each function is adequately documented. This includes information about what the function does, its inputs, outputs, return values, errors, etc. This makes it easier for others (or future you) to understand what the function is supposed to do.
5. **Code Review and Testing:** Have the script reviewed by another pair of eyes and tested in various environments to catch any potential problems early.

3.0.33 `_get_existing_zpool_name`

Contained in `node-manager/rocky-10/zpool-management.sh`

Function signature: cc41e1bc5b39a7a3e09eef4edc9166c5f5783d182d1cd860936367173015ae84

3.0.34 Function Overview

The `_get_existing_zpool_name` function is a bash script intended to fetch the name of the existing zpools (pool of storage devices in ZFS [Zettabyte File System] system) if any on a remote host and then process selection, disk checking and pool creation on the basis of certain conditions and arguments. It also provides the functionality to apply default settings and persist changes.

3.0.35 Technical Description

- **Name**: `_get_existing_zpool_name`
- **Description**: This Bash function fetches an existing pool name from the remote host, executes a series of checks to decide whether a new pool is required or not. It also creates new pools and applies settings depending upon user input.
- **Globals**: N/A
- **Arguments**:
  - `--strategy`: Decides the strategy for disk selection; “first” to choose the first available disk; “largest” to select the largest disk.
  - `--mountpoint`: Specifies the mount point.
  - `-f`: Force a particular action.
  - `--dry-run`: Runs everything as normal, but does not make any changes.
  - `--no-defaults`: Does not apply default settings.
- **Outputs**: Logs
- **Returns**: Value depending upon the success or failure of the various operations within the function.
- **Example Usage**: `_get_existing_zpool_name --strategy largest --mountpoint /mnt/pool`

3.0.36 Quality and Security Recommendations

1. Implement detailed error handling: The function should be able to catch and handle potential edge cases and errors in a more detailed manner to provide better user experience.
2. Increase function modularity: Splitting large functions into smaller, more specific modules or functions could enhance readability, usability, and testing.
3. Secure sensitive data: If this function is being used in a production environment, special care should be taken to secure sensitive or confidential data. For instance, sanitizing inputs and protecting any logged data.
4. Validate user inputs: Function should have detailed checks to validate the user inputs in order to avoid any malicious actions.
5. Enhance logging mechanisms: Using robust and extensive logging mechanisms to identify and troubleshoot the issues quickly.

3.0.37 `_get_partition_device`

Contained in `node-manager/alpine-3/+INSTALLING/installer-functions.sh`

Function signature: ad9bf3c98b29acbfb3b9f9599cb8bbe96ebe1077f8e364763a9d6f603ee239f5

3.0.38 Function overview

The provided Bash script appears to perform disk partitioning operations by determining if the disk is a specific NVMe device and handling RAID related operations if necessary. The `_get_partition_device` function is used for obtaining partition device names. After that, a RAID setup is optionally created using mdadm, depending on the `raid_mode` variable. Successful RAIDs and single disks have their device paths stored in the `boot_device` and `root_device` variables. These paths are then stored to `host_config` using `n_remote_host_variable` function, followed by logging the details into a remote log.

3.0.39 Technical description

- Name: `_get_partition_device`
- Description: A utility function that helps generate correct device names for partitions, especially accurately handling NVMe device names.
- Globals:
  - `raid_mode`: Determines RAID setup mode.
  - `disks`: Array holding the disk details.
- Arguments:
  - `$1`: Represents the disk whose partition is to be identified.
  - `$2`: Represents the partition number on the disk.
- Outputs: Echoes the correct partition device name.
- Returns: Doesn't explicitly return a value. However, `echo` statements implicitly become the return values which can be captured in command substitution (`$()`).
- Example Usage:

```
local disk1_partition_2=$(_get_partition_device "$disk1" 2)
```

3.0.40 Quality and security recommendations

1. The script globally handles multiple disks, partitions and other variables. Use of more localized scope for variables, when possible, would enhance the code quality.
2. The return codes such as '3' and '1' are hardcoded. It's a better practice to use meaningful constant variables for return codes.
3. Errors are logged with related messages but those parts of the script don't exit or stop. Depending on severity, the script could stop at errors.
4. Check for the existence of the required commands like `mdadm` at the beginning of the script for better practice.
5. For security, the script could incorporate additional checks or user prompts before creating, modifying, or deleting disks or partitions.

6. All input and output operations should be validated and sanitized to prevent potential security risks.

3.0.41 `get_src_dir`

Contained in `node-manager/alpine-3/TCH/BUILD/run_osvc_build.sh`

Function signature: 398dd423edabfdc73222dbbf0132c50cff244bac1def7cbd3a1ba846af93b177

3.0.42 Function overview

The `get_src_dir` function is a Bash function designed to output a static string (`"/srv/build/opensvc-om3-src"`) when called. This function is usually utilized to provide a fixed directory path when building or working with the 'opensvc-om3' part of a server structure.

3.0.43 Technical description

- **Name:** get_src_dir
- **Description:** The function echoes out a predefined static string denoting the source directory of 'opensvc-om3'.
- **Globals:** None
- **Arguments:** None
- **Outputs:** This function outputs a static string `"/srv/build/opensvc-om3-src"`
- **Returns:** As the only operation carried out in the function is `echo`, the function does not have a return value. The output string can be considered as its return.
- **Example usage:**

```
$ ./my_script.sh
[... script contents...]
source_dir=$(get_src_dir)
echo $source_dir
```

This will output: `/srv/build/opensvc-om3-src`

3.0.44 Quality and security recommendations

1. If the function is intended to provide a static directory path, consider using a constant variable instead. It's more readable and efficient.
2. If the function might need to provide different directory paths under certain circumstances, it might be better redesigning this function to accept arguments to provide dynamic paths.
3. As this function is returning paths which can be used later in the script for file operations, always ensure the server structure doesn't allow unauthorized access to the file system. Protect these directories with correct permissions and ownership to avoid any potential security issues.

4. It's always good to check if the directory exists before it's used. Add error handling mechanism in the script to handle situations when the directory is not found.
5. Ensure proper usage of this function. Any misuse in file operations can lead to potential data loss or data breaches.

3.0.45 `_log`

Contained in `node-manager/rocky-10/zpool-management.sh`

Function signature: f7f5d16961fc9339682891b49f2fefad3611a12964b1a3cd095d11a46c108959

3.0.46 Function overview

The `_log` function is a logging function that allows us log messages remotely and also locally. It accepts multiple parameters and formats them into a stringified message which is then logged. If the `LOG_ECHO` environment variable is set to `1` (which it is by default), it also prints the log message to the console. It's primarily designed to log operations related to the `zpool_create_on_free_disk` function.

3.0.47 Technical description

- **name**: `_log`
- **description**: This is a logging function. It manages logging the operations pertaining to the `zpool_create_on_free_disk` function. The function does not only log the message to a remote location but also, depending on whether `LOG_ECHO` is set to `1`, echoes the message as an output.
- **globals**: [`LOG_ECHO`: determines whether log messages are also printed to the console, 1 = yes and 0 = no]
- **arguments**: [`$*`: log message parts which are concatenated into a single string]
- **outputs**: This function either logs message remotely or both logs remotely and echo's it locally.
- **returns**: This function does not return any specific value.
- **example usage**:

```
_log "Creating zpool on free disk" "Started"
```

3.0.48 Quality and security recommendations

1. Ensure that the remote log system being used is secured and log data is transmitted over a secure channel
2. Avoid logging sensitive data that could potentially be exploited if logs were to be accessed by unauthorized users

3. Check to ensure LOG_ECHO values are not altered unintentionally which may lead to unexpected functioning
4. It could be useful to apply a standard format to the log messages to make them easier to parse and analyze
5. Make sure that errors in the logging system itself are handled properly to prevent crashes or loss of data
6. Rate-limiting could be beneficial if too many logs are being sent at once, which could pose performance issues.

3.0.49 n_auto_load_network_modules

Contained in `lib/node-functions.d/alpine.d/network-module-load.sh`

Function signature: 39c587b1a3f204bbc660d2ad9160001ec59b462279fb343dd49493da1d02c50f

3.0.50 Function Overview

The Bash function `n_auto_load_network_modules()` is designed to scan and identify PCI devices that require network controllers and kernel modules associated with them, common network modules required based on hardware, and dependent firmware through the system messages. If necessary, it automatically loads these modules and firmware, and persistently adds them to the system. It logs every action taken, making it easy to trace back any changes made to the module list.

3.0.51 Technical Description

3.0.52 n_auto_load_network_modules_safe

Contained in `lib/node-functions.d/alpine.d/network-module-load.sh`

Function signature: 2d17eac72861b9c276bcd9107ac275f8ea79ede60dea8fe3e04633b9e8f68025

3.0.53 Function overview

This function, `n_auto_load_network_modules_safe()`, automates the process of loading necessary network modules. It starts by checking the availability of the required kernel modules with the help of the `n_ensure_modules_available` function. If the modules aren't available, it logs an error message "[NET] Cannot load network modules - kernel modules not available" and returns 1, indicating failure. Subsequently, if the function hasn't returned due to missing modules, it proceeds to load the network modules.

3.0.54 Technical description

```
- **name**: `n_auto_load_network_modules_safe`
- **description**: This function automates the loading of network
 ↪  modules. It checks if the necessary kernel modules are
 ↪  available for the operation. If not available, it logs an
 ↪  error message and returns '1'. If the essential modules are
 ↪  successfully found, it then initiates the loading of network
 ↪  modules.
- **globals**: None
- **arguments**: None
- **outputs**: It either logs a failure message stating unavailable
 ↪  kernel modules or proceeds to load the network modules.
- **returns**: Returns `1` if kernel modules aren't available
 ↪  indicating a failure. It doesn't have a clearly defined
 ↪  success return value because the function's successful
 ↪  execution results in the networking modules being loaded.
- **example usage**:
    ```bash
    n_auto_load_network_modules_safe
    ```
```

3.0.55 Quality and security recommendations

1. Consider designing the function to return a success value (0) or more descriptive error codes for different kind of errors, which can be helpful in troubleshooting.
2. Avoid logging vaguely worded error messages. Instead, include more detailed and specific messages that describe exactly what went wrong.
3. For better security, consider checking for elevated permissions (root) considering this function deals with loading network modules.
4. Given the critical functionality of this command, ensure that it's protected from injection attacks, possibly via rigorous input validation.
5. Always keep the function up to date with any changes in the kernel or networking modules.

3.0.56 `n_build_apk_packages`

Contained in `lib/node-functions.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: 59061ba339c11e34640c2aa719aaf1ae197da63bc619460a88b52b9863412ca2

3.0.57 Function overview

The Bash function `n_build_apk_packages` is a script that automates the process of building APK packages for an OpenSVC server and client. It ensures that required environment variables are set, verifies package directories exist, then proceeds to generate checksums, keys, and builds the actual APK packages. If an error occurs at any

step, the function returns an error message and exit code of 1, halting the process. Upon successful completion, it copies the built APK packages to the specified directory, logs a success message, and returns an exit code of 0.

3.0.58 Technical description

- **Name**: `n_build_apk_packages`
- **Description**: This function is utilized for building APK packages for the OpenSVC server and client automatically.
- **Globals**: [`OPENSVC_SERVER_PKG_DIR`: The OpenSVC server package directory location, `OPENSVC_CLIENT_PKG_DIR`: The OpenSVC client package directory location, `OPENSVC_PACKAGE_BASE_DIR`: The directory to copy the completed packages to, `OPENSVC_VERSION`: The version of OpenSVC]
- **Arguments**: No arguments expected.
- **Outputs**: Echoes status messages and errors to the console. If successful, it builds and copies APKs to the specified directory.
- **Returns**: Returns `1` when it encounters an error (environment variable unset or directory/package not found) or `0` upon successfully building and copying the APKs.
- **Example Usage**:

```
n_build_apk_packages
```

3.0.59 Quality and Security recommendations

1. Always use double quotes around variable substitutions to avoid word splitting and pathname expansion.
2. Each environment variable should preferably have its existence checked at the beginning to ensure they are set.
3. The change of directory (cd) without checking if the destination exists or is a valid directory may cause unforeseen errors. Always double-check the availability and validity of the directory before changing into it.
4. Abuild commands are run ignoring the exit code, which might hide potential errors or issues during the packaging process. Consider handling the exit code of each critical function in a more secure way.
5. In the cp commands, always check if source and destination directories are valid before proceeding. An error prompt will help pinpoint any issues with the script.

3.0.60 `n_build_opensvc_binaries`

Contained in `lib/host-scripts.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: 7cc90027c9d9a353b0cd467113ad5b2a624efc0d5c66c5eb4a1ce7cbd73ec71b

3.0.61 1. Function Overview

The function `n_build_opensvc_binaries` is a shell script that builds the OpenSVC binaries. It checks certain environment variables for the build directory and OpenSVC version, and ensure the build directory exists. If the checks fail, the build process stops and returns an error. Upon successful verification of the build directory environment, the function navigates to the build directory, manages the git ownership, runs the make process, and checks and verifies the binaries. Once binaries are verified, their executability is ensured and the function returns a success status.

3.0.62 2. Technical Description

- **Name:** `n_build_opensvc_binaries`
- **Description:** The function is aimed at building OpenSVC binaries. It checks for environment variables, navigates to the correct directory, manages git directories, runs the make process and verifies the binaries. On successful completion, it ensures the executability of binaries and logs the successful build.
- **Globals:** `[ OPENSVC_BUILD_DIR: Directory for OpenSVC build, OPENSVC_VERSION: The version of OpenSVC for building binaries ]`
- **Arguments:** `[ None ]`
- **Outputs:** Messages indicating status of building process or error messages on unsuccessful operations.
- **Returns:** The function returns `1` in case of an error and `0` if the build process completes successfully.
- **Example Usage:** `n_build_opensvc_binaries`

3.0.63 3. Quality and Security Recommendations

1. Better error handling could be implemented for the process where environment variables are checked. If they are not set, this could still be resolved within the script rather than returning an error.
2. Ensure proper permissions for the build directory. Restrict unauthorized users from accessing the build directory to prevent malicious tampering with the binaries.
3. Implement proper logging mechanism. Currently, the error messages are only echoed and not logged which could make debugging issues harder in the future.
4. Ensure user who is calling the script has adequate permissions to avoid potential permission issues.
5. Implement checksum verification after binaries are built to ensure the integrity of the binaries.

3.0.64 n_build_opensvc_package

Contained in `lib/node-functions.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: 4ff11e24efab635ad3e7b651cc12ccd1498b33c83c50ee3968dd9e66651b9a8a

3.0.65 Function overview

The Bash function `n_build_opensvc_package()` is a script to automate the build process for the OpenSVC agent on Alpine Linux. This script automatically handles version specification, repository cloning, checking network connectivity and compatibility, build dependencies, and performs build cleanup unless otherwise specified.

3.0.66 Technical description

Name: `n_build_opensvc_package`

Description: This function automates the entire build process for OpenSVC agent in Alpine.

Globals:

- `OPENSVC_VERSION`: Defines the opensvc version
- `OPENSVC_GIT_TAG`: Specifies the git tag to checkout from OpenSVC repository
- `OPENSVC_BUILD_DIR`: Specifies the directory where the build will take place

Arguments:

- `$1`, `$2`: The arguments to the function serve to provide additional options, including specification of alpine version, opensvc git tag, whether to keep the build directory, or to request help.

Outputs:
The function outputs status messages for each step of the build process, error messages when applicable, and success messages upon successful completion of the build.

Returns:
The function will return 1 in an error state if any of its steps fail to properly execute, and 0 upon a successful completion.

Example Usage:

```
n_build_opensvc_package --alpine-version 3.14 --om3-version v1.9.1 --keep-build
```

3.0.67 Quality and security recommendations

1. **Double check all command line arguments**: To avoid any potential vulnerabilities or crashes, ensure command line arguments are thoroughly checked and sanitized before use.

2. **Check for potential failure states**: Ensure every command that has the potential to fail has a corresponding error check. This includes not only the steps in the build process, but any file or directory manipulation.
3. **Implement more robust logging**: Consider creating separate log files for the build process, which could be useful for troubleshooting in the event of a failure.
4. **Enforce permissions control**: To prevent potential unauthorized access or modifications, ensure correct permissions on directories and files that the script interacts with.
5. **Consider implementing checksum verification for downloads**: Adding checksum verification for any downloads would enhance the security of the script by verifying the integrity of the download before it is used.

3.0.68 `n_check_build_dependencies`

Contained in `lib/node-functions.d/alpine.d/BUILD/01-install-build-files.sh`

Function signature: a33ac47a40942c3b9794baf64f5d9947b4880122bbe208e0b9290e0179c6e822

3.0.69 Function Overview

This bash function `n_check_build_dependencies` is designed to verify the presence of build dependencies required by software before its installation. It does this by looping through a local array `deps` that stores the dependencies ("command:package").

If a dependency is present, it gets echoed to the console with a check mark [✓]. If a dependency is missing, it gets echoed with an "x" mark [☒] and is added to a list of missing packages. If any package is missing, the function prompts the user on how to install the missing packages using `apk add`. The function then logs the missing dependencies remotely.

3.0.70 Technical Description

- **Name:** `n_check_build_dependencies`
- **Description:** Checks if the necessary build dependencies are already installed on the system.
- **Globals:** None
- **Arguments:** None
- **Outputs:** Echoes the status of the build dependencies check
- **Returns:** 1 if some dependencies are missing; otherwise returns 0
- **Example Usage:**

  ```
  n_check_build_dependencies
  ```

The command above will check for the presence of the required dependencies and echo their status.

3.0.71 Quality and Security Recommendations

1. Consider adding argument input validations, ensuring that the inputs to the function are in the expected format and type.
2. It's advisable to use long flags (`--long-flag`) instead of short flags (`-l`) in scripts as they are more readable and self-descriptive.
3. Incorporate error handling to handle failed conditions, such as if the `command` command itself fails or returns unexpected output.
4. Avoid globally accessible (read, write, execute) files or directories. This will reduce the attack surface.
5. Consider the chances of a command injection, where an attacker could manipulate variables to execute arbitrary commands on the system. Sanitize any inputs that your scripts receive.
6. Regularly update and patch your software to minimize the risk of security vulnerabilities.

3.0.72 `n_check_go_version_compatibility`

Contained in `lib/node-functions.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: 256910e2851d59c378f8e75bc1cc04adab0059a15b36c52bc53b5dcb82e9a2b9

3.0.73 Function Overview

The function `n_check_go_version_compatibility` checks the compatibility between the installed version of Go and the required version as specified in the Go module file (`go.mod`) of a given repository. Given a git tag, it navigates to the source directory and fetches the necessary version from the `go.mod` file. It also retrieves the installed version of Go for comparison. If the installed version is not below the required one, it declares the two as "Compatible"; otherwise, it returns an error.

3.0.74 Technical Description

- **Name:** `n_check_go_version_compatibility`
- **Description:** This function checks if the installed Go version is compatible with the version required by the Go project.
- **Globals:** None.
- **Arguments:**
  - `$1: git_tag`. Represents the specific git tag to be fetched from the repository.
  - `$2: source_dir`. The directory of the source code.

- **Outputs:** Prints out the status of Go version compatibility (Required, Installed, and Status) along with specific error or warning messages.
- **Returns:** The function returns `0` on successful execution and compatible Go versions. It returns `1` in case of errors or incompatible versions.
- **Example Usage:** `n_check_go_version_compatibility v1.0.0 src/`

3.0.75 Quality and Security Recommendations

1. Validate the inputs, especially the `git_tag` and `source_dir`. Add checks to ensure that they are not empty, null, or potentially harmful content.
2. Sanitize all output. This will prevent any potential output-related security risks.
3. Implement logging for debugging purposes. This will help to keep track of any issues that occur during the execution of the function.
4. Handle all possible edge cases, such as handling a version string that does not follow the expected format. This will prevent the function from behaving unexpectedly.
5. Consider further improving error handling. At the moment, any warning or error simply prints a message and returns an exit code. It could be useful to throw exceptions in certain cases, to provide more contextual information about the error.

3.0.76 `n_clone_or_update_opensvc_source`

Contained in `lib/node-functions.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: cdf629e03b8f345fb7a13c0b03de61c64e3c3c8cf1d6cedc7b8a88377b22a57a

3.0.77 Function overview

The `n_clone_or_update_opensvc_source` function is designed to handle the source directory for the OpenSVC software package. The function checks if the source repository exists or not, and if it does, updates the repository from the remote server. If the repository does not exist, the function attempts to clone it from the repo_url (`https://github.com/opensvc/om3`). The function makes use of git commands and basic directory manipulation commands such as mkdir.

3.0.78 Technical description

- Name: `n_clone_or_update_opensvc_source`
- Description: Manages the source directory for the OpenSVC package, updating it if it exists or cloning the repository if it does not.
- Globals: None
- Arguments:
  - None

- Outputs: Various status and error messages related to the process of updating or cloning the repository.
- Returns:
  - 0: The function succeeded in either updating or cloning the repository.
  - 1: The function encountered an error in either updating or cloning the repository.
- Example Usage:

```
source_dir="$(get_src_dir)"
n_clone_or_update_opensvc_source
```

3.0.79 Quality and security recommendations

1. Consider adding more detailed error handling, to give the user a more specific idea of what might have gone wrong in the event of a failure.
2. Explicitly validate and sanitize any user-provided input that is used to form the directory paths. This not only avoids the risk of command injection attacks but also helps prevent accidental misconfiguration by the user.
3. Make use of variables to store reusable command or path strings, which not only makes the script more maintainable but can also help to avoid typing mistakes or inconsistencies in command usage.
4. Consider logging more detailed operation feedback to a dedicated log file for easier debugging.
5. Always check the return value of system and external commands to ensure they have executed successfully before proceeding.

3.0.80 `n_configure_bash_shell`

Contained in `node-manager/base/n_shell-config.sh`

Function signature: c83730f9e26d12f2581923a5fe9680453330d4ab55e3ed697ef18a019bf3a73e

3.0.81 Function overview

The `n_configure_bash_shell()` function facilitates seamless bash shell configuration as a preferred shell. The function initiates with a status log styled as "[INFO] Configuring bash as default shell". The function checks whether bash shell is installed and then proceeds to change the /bin/sh symlink from its current target to bash. Once the symlink has been successfully configured to bash, the function creates a profile.d drop-in feature for additional versatility in command executions. The function provides an informative output log if the bash shell configuration is completed successfully, and returns status codes depending on the execution status.

3.0.82 Technical description

- **Name**: n_configure_bash_shell
- **Description**: Configures bash as the default shell, checks if the bash shell is installed, modifies the /bin/sh symlink from its current state to bash, and finally creates a profile.d drop-in file in /etc/profile.d/.
- **Globals**: None.
- **Arguments**: None.
- **Outputs**: Status and error messages are outputted to stderr to inform the user about the current status of the function.
- **Returns**: The function primarily returns three codes upon its completion:
    1. Returns 0 if the function execution is successful.
    2. Returns 1 if there is an error in either creating the bash symlink or if the bash shell is absent.
    3. Returns 2 if the function fails to either create a profile drop-in or set permissions to it.
- **Example Usage**:

```
n_configure_bash_shell
```

3.0.83 Quality and security recommendations

1. Include more conditional statements to check other possible points of failure, such as checking if the current shell is already bash.
2. Handle possible exceptions such as the absence of some files or directories the function depends on.
3. Encapsulate the global variables used within the function to ensure there would be no conflicts with other scripts.
4. Develop a rollback mechanism that reverts any changes made if the function execution fails at any point.
5. The error messages should be more descriptive to better assist in troubleshooting.
6. User input should be validated and sanitized to mitigate the possibility of code injection.
7. Input parameters should be handled properly to improve the function's reusability.
8. Log files should be maintained to keep track of every operation for auditing and troubleshooting purposes.
9. Ensure the script is running with minimal permissions to reduce possible risks.

3.0.84 `n_configure_ips_profile`

Contained in `lib/node-functions.d/alpine.d/alpine-lib-functions.sh`

Function signature: c2867bf74734c8e64d8346a86e77bca65505b85d377cf41b23125a37a59d5e4f

3.0.85 Function overview

The Bash function `n_configure_ips_profile()` creates or reconfigures a HPS profile script located in "/etc/profile.d/hps-env.sh". The purpose of this function is to ensure that the HPS environment setup executes for login shells. If the specific directory does not exist, the function will create it. This function also contains error and success logging functionality.

3.0.86 Technical description

This function will have the following properties:

- **Name:** `n_configure_ips_profile`
- **Description:** Creates or alters a HPS profile script.
- **Globals:** None.
- **Arguments:** This function does not take any arguments.
- **Outputs:** Logs either a success message stating that the profile script has been created successfully, or an error message indicating that script creation failed.
- **Returns:** Returns 1 if creation of the profile script fails, otherwise returns 0 (success).
- **Example Usage:**

  ```
  n_configure_ips_profile
  ```

3.0.87 Quality and security recommendations

For the improvement of according function's quality and security the following steps are recommended:

1. Implement detailed logging, including timestamps, which may help troubleshoot potential issues.
2. Incorporate error checking after each critical step, not just the creation of the profile script.
3. Replace hard-coded file paths with either configurable settings or variables. This will help to avoid accidental deletion or modification.
4. Include additional checks to ensure the system user has sufficient permissions before trying to change anything.
5. Implement some kind of version control for the modified scripts to prevent potential loss of important changes.
6. Ensure to limit the scope of environment variables and sensitive data, not to disclose inadvertently.

3.0.88 n_configure_minimal_networking

Contained in lib/host-scripts.d/alpine.d/networking-functions.sh

Function signature: 905f4f54d7e9ce8973823b4bbf269c4358c4830871e5290d9228cc2a31beed11

3.0.89 Function Overview

The function n_configure_minimal_networking configures minimal networking for a Linux machine. It is particularly aimed at enabling the service dependencies to initialize and operate properly.

The function first collects networking details such as the principal interface, IP address, netmask, and gateway for the Linux machine. A loopback interface is then brought up if not yet up since it is critical for many services. It then ensures that networking is included in the system boot runlevel to support future machine restarts. Finally, if the networking service is not running already, an attempt is made to start it.

3.0.90 Technical Description

- **name:** n_configure_minimal_networking
- **description:** Configures minimal networking for a machine ensuring service dependencies are met. It sets up the primary network interface, gateway, and adds networking to boot runlevel. If networking service is not running, it is started.
- **globals:** None.
- **arguments:** None.
- **outputs:** Creates a '/etc/network/interfaces' file with the appropriate network configuration, and commands to start networking service in the console.
- **returns:** If successful, exits with a 0 status code. If the networking service can't be started, it exits with a status code of 1.
- **example usage:** bash n_configure_minimal_networking

3.0.91 Quality and Security Recommendations

1. Secure the route information - The function currently extracts the gateway directly from the routing table. An attacker might manipulate this information leading to a potential security issue. Validate and sanitize the input consumed from the routing table to guard against injection attacks.
2. Enhance error handling - Right now, if there's any failure, error handling is quite generic. Specific and clear error messages should be returned for each failure scenario to aid in troubleshooting.
3. Secure temp file - The function writes network configuration to '/etc/network/interfaces'. Any potential security issues associated with file permissions, access rights, and data within the file should be well handled.

4. Consider using more secure coding practices like hardcoded values (for netmask, for example) can be replaced with queries to relevant resources.
5. Include logging functionality - To ensure traceability and easier debugging, consider including logging functionality that provides an audit trail of actions performed by this function.
6. Validate network state before configuration - To avoid unnecessary cycles and operations, check the network state before proceeding with the configuration. This will also avoid overlapping configurations which may lead to errors or unpredicted behaviors.

3.0.92 `n_configure_motd`

Contained in `lib/host-scripts.d/alpine.d/console-control.sh`

Function signature: d87371da5d1ef4cf5f7025f69387b9e18a089f696a2ece496bc5166f820d9398

3.0.93 Function Overview

The function `n_configure_motd()` is used to configure the Message Of The Day (MOTD) with information about the node on a Unix/Linux system. The MOTD is a brief message that users see when they log into a system. This function generates a dynamic MOTD that displays node information on user login. It also creates a static MOTD where node information is stored.

3.0.94 Technical Description

- Name: `n_configure_motd`
- Description: This function configures the MOTD with node information. It first creates a script with the name `hps-motd.sh` in the directory `/etc/profile.d/` that generates a dynamic MOTD. The dynamic MOTD shows node information on a user login. The function further creates a static MOTD where it stores the node information. Finally, the function logs that the MOTD has been configured.
- Globals: None
- Arguments: No arguments are passed to this function.
- Outputs: This function does not explicitly output any value to the stdout, it updates and creates system files `/etc/profile.d/hps-motd.sh` and `/etc/motd`, and logs messages using `n_remote_log` function.
- Returns: Always returns 0 indicating successful execution.
- Example usage: To configure the MOTD with node information, you can call the function as follows `n_configure_motd`

3.0.95 Quality and Security Recommendations

1. All global variables and constants should be defined at the beginning of the script to improve readability and ease of maintenance.
2. Always define the directory paths and filenames as variables at the beginning of the script. It's especially useful when several functions are part of the same script or if you expect to reuse the function in different contexts.
3. Use the bash `set -e` option at the beginning of your script to make sure the script exits whenever any command it runs exits with a non-zero status.
4. Make sure that permissions of MOTD scripts and static files are appropriate and do not allow unauthorized access. The default permissions set by `chmod +x` might not always be the best choice.
5. Always log successes and errors in your function to a log file. Using `n_remote_log` function seems to be a good choice.
6. In most Linux distributions, you may need root/sudo privileges to make changes in `/etc/motd` or `/etc/profile.d/`. Always use caution to avoid overwriting crucial system files.
7. Make sure that scripts and commands are shielded against injection and other types of attacks.

3.0.96 `n_configure_persistent_network`

Contained in `node-manager/alpine-3/+INSTALLING/installer-functions.sh`

Function signature: 2fabe8102809acae470b9ae82768d6bb8abc3f68e229cef1af7a163c7b85fec8

3.0.97 Function Overview

The `n_configure_persistent_network()` function is used to manage the network configuration in a persistent manner. This function takes a target root directory as an argument and configures the network interfaces within that directory. It initiates this by ensuring the directories exist, setting the management interface to `eth0` and logging the action. The function then creates network interface files with DHCP configuration. If these actions are not successfully executed, the function will log an error and return an echo code of 2. However, if all operations are successful, it will enable the networking service in the default run level, log the action and return a `0` echo code indicating success.

3.0.98 Technical Description

- **Function Name:** `n_configure_persistent_network()`
- **Description:** This bash function configures the network in a persistent manner on a target root directory specified as an argument.
- **Globals:** None

- **Arguments:** [`$1 (target_root)`: A string showing the path of the target root directory. It serves as the root directory to which network configuration will be performed.]
- **Outputs:** Logs informative, debugging, and error messages during the execution.
- **Returns:** `0` if the function successfully creates an interfaces file and enables networking service. If creating the interfaces file fails, it returns the code 2.
- **Example Usage:** `n_configure_persistent_network "/target/root/directory"`

3.0.99 Quality and Security Recommendations

1. It would be beneficial to ensure that the target_root passed as an argument is always a valid directory; this can be achieved by adding error checking validation in the function.
2. While the function does a good job of logging actions, it can still be enhanced by adding more logging in complex actions.
3. Since this function handles files and directories, ensure that the permissions for these files are tightly controlled to prevent unauthorized access or modifications.
4. Instead of hardcoding interface names like `eth0`, it is recommended to pass these as arguments to make the function more general and flexible.
5. Although the function handles error situations well, it could be improved by managing different types of errors uniquely, thereby improving the fault tolerance and robustness of the overall system.

3.0.100 `n_configure_reboot_logging`

Contained in `node-manager/alpine-3/alpine-lib-functions.sh`

Function signature: 1572809279fe0aeae1f526164db98f88918aa31dd7a922a93277fed30cf125dc

3.0.101 Function overview

The function `n_configure_reboot_logging` is primarily designed to handle logging for system reboot procedures. It begins by ensuring Bash is available and that the required directories exist. The function then fixes any broken symlinks, creates wrapper scripts for various commands (e.g. reboot, poweroff, halt), and manages shutdown logging. The execution of the function concludes by updating the system PATH and enabling the local service, if not already enabled. For each key step, appropriate messages are logged either remotely or locally, providing essential visibility to system administrators.

3.0.102 Technical description

Here is a technical breakdown of the `n_configure_reboot_logging` function:

- **Name**: `n_configure_reboot_logging`

- **Description**: This function is designed to handle and configure reboot logging in a system. It sets up a process to ensure appropriate logging for reboot, poweroff, and halt commands. Further, it deals with shutdown procedures and updates the system profile accordingly.
- **Globals**: None.
- **Arguments**: None.
- **Outputs**: Logs of actions such as configuration of reboot logging, error messages, successful configuration of reboot logging, among others.
- **Returns**: The function will return 1 when bash is not found, and in the case of successful execution, it returns 0.
- **Example usage**: N/A (This function would typically be called directly within a system script without arguments)

3.0.103 Quality and security recommendations

1. The function should validate the existence of all required directories and files for the logging process.
2. Ensure that the system has appropriate permissions for file and directory operations.
3. Implement error handling for function calls and command executions.
4. Regularly audit the logs to track and respond to any unexpected behavior or errors.
5. Avoid disclosing sensitive information in logs, use anonymization where required.

3.0.104 `n_configure_syslog`

Contained in `lib/node-functions.d/alpine.d/configure-syslog.sh`

Function signature: 1b14ccc8b598f0508987ad09313249bf7fcd60c8f4aeff98f6f143b369e02ff2

3.0.105 Function Overview

The function `n_configure_syslog()` is a bash script designed to configure and start a syslog service on the host system. It accomplishes this by writing configuration details to the `/etc/conf.d/syslog` file, ensuring the service will start on system boot using `rc-update` and then manually starting the service with `rc-service`. The function also sends a test log message and then verifies that the syslog service has been configured and started. It takes in one optional argument representing the IP address of the host.

3.0.106 Technical Description

In terms of technical description, the function can be defined as follows:

- Name: `n_configure_syslog`

- Description: This function configures and starts the syslog service on a host machine.
- Globals:
  - VAR: No global variables are manipulated or altered in this function.
- Arguments:
  - $1: This optional argument represents the IP address of the host machine. The default value is '10.99.1.1'.
- Outputs: Configures syslog service, starts it, and verifies its configuration and start.
- Returns: Does not return a value.
- Example Usage: `n_configure_syslog 10.88.1.1`

3.0.107 Quality and Security Recommendations

1. Basic error handling should be added to ensure the function's execution in conditions like absence of required permissions or in case of any other unforeseen errors.
2. Logging configurations and operations should be kept secure and confidential to prevent any security breach.
3. It is recommended to include validation checks for the input parameters to ensure they have the expected format.
4. The function should be tested for all edge cases and potential exception scenarios to ensure its robustness.
5. Additional comments could be included to improve readability and maintainability of the function.
6. Implement a method to restart the syslog service if the host reboots.
7. For better security measures, all the written logs should be encrypted whenever necessary.

3.0.108 `n_console_message`

Contained in `lib/host-scripts.d/pre-load.sh`

Function signature: 36e26683aca03d37f7320e5524aa23849cd906d839f1064363a2204b560d0b17

3.0.109 Function overview

The `n_console_message` is a bash function that utilizes conditional statements to print messages to the console. The messages are customizable, with a default system message in the case no input message is provided.

3.0.110 Technical description

- **Name**: n_console_message

- **Description**: The function is designed to take in a user-inputted message, defaulting to a system message if none is provided. It checks the write abilities to both the /dev/console and the /dev/tty1, if accessible, the message is printed to both.
- **globals**: None.
- **Arguments**: [$1: This represents the user-inputted message. If this argument is missing the function will use a default 'System message'.]
- **Outputs**: The function may output a user-inputted message or the default 'System message' if the console and tty1 are both available and writable.
- **Returns**: The function returns 0 indicating successful completion.
- **Example Usage**:

```
n_console_message "This is a message"
```

Above will print [HPS] This is a message in the /dev/console and /dev/tty1 if they are writable.

3.0.111 Quality and security recommendations

1. Always validate whether the user-inputted message is in the desired plain text format to avoid any chance of code injection.
2. Additional error checking mechanisms can be included in case the console or tty1 are not writable.
3. It would be beneficial to include custom exit statuses for different failure instances, instead of mere success status. This will aid in easier debugging.
4. Regular reviews and updates should be performed to ensure that all security protocols are met.

3.0.112 `n_create_apk_package_structure`

Contained in `lib/node-functions.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: 26a2bf4a08a3b2e3efb1f9c8b0b1e77027a4f1e8ec35c925f6214768f4107b95

3.0.113 Function overview

The function `n_create_apk_package_structure` is a bash function primarily used for creating APK package structures for a software called OpenSVC. It checks for environment variables, verifies if necessary binaries exist, detects the version of Alpine, and initializes directories and files needed for the package structure of OpenSVC. It also includes the creation of OpenRC init script.

3.0.114 Technical description

- **Name**: `n_create_apk_package_structure`
- **Description**: This script intends to create APK package structures for OpenSVC. It performs several checks before proceeding to create directories and files.

- **Globals**: [OPENSVC_VERSION: Current version of OpenSVC, OPENSVC_BUILD_DIR: Directory where OpenSVC binaries are located]
- **Arguments**: The function takes no arguments.
- **Outputs**: The function outputs string messages indicating the status of APK package structure creation. It informs about OPENSVC version, APK version, Alpine version, and the package directory details.
- **Returns**: The function may return 1 if any of its checks fail. This indicates a failure in creating the APK package structure due to missed requirements.
- **Example Usage**: `n_create_apk_package_structure`

3.0.115 Quality and security recommendations

1. Consider validating the format of the version being read from environment variable OPENSVC_VERSION. This prevents errors related to invalid versions.
2. Check the existence of "/sbin/openrc-run" before writing to it.
3. Handle possible permission errors while creating new directories or copying om binary to its destination.
4. Please use more specific error messages for each case where the function might return 1. For example, message whether the environment variable wasn't set or the om binary wasn't found.
5. Update the script to not default to a predefined Alpine version if it cannot detect the current version. Instead, consider alerting the user about the necessity to provide this information.

3.0.116 `n_create_cgroups`

Contained in `lib/node-functions.d/alpine.d/alpine-lib-functions.sh`

Function signature: 74017b13a50b229e489bff7300d12a2272b33d7683dd3c299e1f94d5c52336e9

3.0.117 Function Overview

The bash function `n_create_cgroups()` configures cgroups v2 in a Linux system. It first checks if cgroup2 is already mounted. If not, the function attempts to mount the cgroup2 filesystem on /sys/fs/cgroup. It then verifies if the mount was successful. If the mount was successful, the function then checks if cgroup2 is added to /etc/fstab. If not, the function adds it to ensure that the cgroup2 filesystem is mounted at system startup.

3.0.118 Technical Description

- **name**: `n_create_cgroups`
- **description**: The function creates and configures cgroups v2.
- **globals**: None
- **arguments**: None

- **outputs**: Logs through `n_remote_log`. Various messages regarding the status of cgroup2 mount and addition to /etc/fstab.
- **returns**: Returns 0 if the function executes successfully. If cgroup2 mount fails, the function returns 1. If there is a failure updating /etc/fstab, it returns 2.
- **example usage**: `n_create_cgroups`

3.0.119 Quality and Security Recommendations

1. Implement error checking to ensure the `n_remote_log` function exists and can be called.
2. Make sure the script is running with the necessary permissions to execute mounting and edit /etc/fstab.
3. Input validation: No input is taken from the user, reducing the risk of input-related vulnerabilities.
4. Implement logging to a separate file to capture the history of cgroup2 configuration.
5. Unmount cgroup2 filesystem before reattempting a failed mount.
6. Avoid hard-coding paths, make the script adaptable by defining path variables.

3.0.120 `n_detect_missing_network_drivers`

Contained in `lib/node-functions.d/alpine.d/network-module-load.sh`

Function signature: 4705c8c63726cec000bec963a72a3b5bc18db52dcdd0090afabaee7b185dfabb

3.0.121 Function overview

The `n_detect_missing_network_drivers` is a function that checks for network devices that do not have associated drivers. It first checks for unclaimed network devices using `lshw` command if available. It then proceeds to check for PCI network devices without drivers using `lshw` command if available. It also tries to identify needed modules for the drivers if possible.

3.0.122 Technical description

- Name: `n_detect_missing_network_drivers`
- Description: A function that checks for network devices, both general and PCI, without associated drivers.
- Globals: No global variables are involved in this function.
- Arguments: No arguments are taken by this function.
- Outputs: Outputs the status of network devices and PCI devices, and any unclaimed devices found.
- Returns: No value is returned since the function only uses `echo` to output to the console.

- Example usage: the function can be called directly `n_detect_missing_network_drivers`.

3.0.123 Quality and security recommendations

1. It is recommended to use local variables as often as possible to avoid potential conflict with global variables or other scripts.
2. For better security, consider checking whether `lshw` and `lspci` commands exist before running this script.
3. The script should be executed with minimal privileges to reduce potential security risks.
4. Best practice dictates adding error handling to handle exceptions such as missing and inaccessible commands, or unexpected input.
5. Regular updates and code reviews can help in maintaining the quality and security of the code.

3.0.124 `n_disable_getty_alpine`

Contained in `lib/host-scripts.d/alpine.d/console-control.sh`

Function signature: 343d39eecf990fe5be10497769cdb8aa77d7ae96ed497cb2761fd51e47cac66c

3.0.125 Function overview

The bash function `n_disable_getty_alpine` is designed to disable the getty service in Alpine Linux, which is used for signing onto a terminal. Getty is disabled to setup a custom console display. The function primarily logs the disabling process, creates a copy of the original `inittab` file for backup, and modifies the `inittab` file to remove getty. It then adds a custom console display script and instructs the init process to reload. Finally, a late startup script is removed as its execution is now handled by `inittab`.

3.0.126 Technical description

Here is a technical breakdown of the function:

- **Name**: `n_disable_getty_alpine`
- **Description**: This function is designed to disable getty and setup a custom console display on Alpine Linux.
- **Globals**: None.
- **Arguments**: None.
- **Outputs**: Modifies `/etc/inittab` to remove getty lines and add custom console display, creates and makes executable a script at `/usr/local/bin/hps-console-display`, and removes a late startup script at `/etc/local.d/99-console-display.start`.
- **Returns**: `0` indicating successful execution.
- **Example usage**: `n_disable_getty_alpine`.

3.0.127 Quality and security recommendations

1. Always ensure that the commands used in the function are safe and intended for the desired outcome.
2. Backup important files, like `inittab`, before making modifications.
3. Monitor the log output to detect any abnormalities during execution.
4. Check for the existence of files and directories before attempting to modify or remove them to avoid runtime errors.
5. Consider potential security implications of disabling getty such as the impact on user terminal login.
6. Use this function with caution, as it permanently modifies system configurations.
7. Ensure to test this function in a controlled environment before using it in a production environment, to avoid unwanted outcomes.

3.0.128 `n_display_info_before_prompt`

Contained in `lib/node-functions.d/common.d/console.sh`

Function signature: dcac048f62180791d143d456e76ba9be0d27cf9a94e052158bdc2101bc2a17cd

3.0.129 Function overview

The bash function `n_display_info_before_prompt()` serves to display basic node information on the console every time it is restarted. The function first logs that the node information is being displayed and then checks whether the console is disabled by scanning through `/etc/inittab`. If the console is disabled, a log about it is created and the display will be handled by init respawn instead. Else, the function makes sure that the `n_node_information` is available and updates the issue file with the relevant node information. This issue file is then displayed on the console providing the node info logs.

3.0.130 Technical description

- **Name**: `n_display_info_before_prompt()`
- **Description**: This function log's the node information, checks if the console is disabled, generates node info logs, and creates updates the issue file to be displayed on console.
- **Globals**: `${console_disabled}`, a binary variable to check if console is disabled or not.
- **Arguments**: No arguments accepted.
- **Outputs**: Logs of node information, sends them to remote logs when necessary, and displays them on the console.
- **Returns**: The function returns 0 indicating successful execution.
- **Example usage**: To use this function, call it without any arguments like so: `n_display_info_before_prompt`.

3.0.131 Quality and security recommendations

1. Handle errors when the `n_node_information` is not available or fails to execute. This would make the function more resilient.
2. Use full paths for all binaries to avoid dependency on PATH that can be exploited in some cases.
3. Restrict editing rights to `/etc/issue` to prevent tampering.
4. Document the requirements and side effects of the function more explicitly.
5. Validate and sanitize any external inputs used in the function to avoid injection attacks.
6. Be cautious about symlink attacks, especially when writing to `/etc/issue`. Make sure that the intended real file gets edited.
7. Implement some form of rate limiting to the function to prevent abuse.

3.0.132 `n_enable_console_output`

Contained in `lib/host-scripts.d/pre-load.sh`

Function signature: e0d095431c3b9587d08666c0618947e69ed05b49e76a2626b8c52d77ed672e8f

3.0.133 Function Overview

The `n_enable_console_output` function is primarily designed to configure the system for console and boot message output. This function verifies the presence of certain files and then modifies or appends key-value pairs therein to alter system behavior. It also activates verbose console output and ensures OpenRC service messages appear on the console.

3.0.134 Technical Description

- **name:** n_enable_console_output
- **description:** This function enables verbose console output and ensures RC messages are displayed at console. It sets the RC_QUIET and RC_VERBOSE globals to 'no' and 'yes' respectively. If /etc/rc.conf file is present, it modifies the 'rc_quiet' and 'rc_verbose' parameters in the file to 'NO' and 'YES' respectively, if they exist. Otherwise, they are appended to the file.
- **globals:** [RC_QUIET: used to control whether RC messages are suppressed on console, RC_VERBOSE: used to control level of verbosity for RC messages]
- **arguments:** No arguments required.
- **outputs:** It outputs the log message "Enabled verbose console output" and "Configured console for boot message output" via the n_remote_log function.
- **returns:** It returns 0 indicating successful execution.
- **example usage:**

```
n_enable_console_output
```

3.0.135 Quality and Security Recommendations

1. Ensure file permissions are correctly set for scripts running this function to avoid unauthorized access.
2. Verify the existence of /proc/sys/kernel/printk and /etc/rc.conf before trying to read/write into it to prevent potential file operation errors.
3. Ensure error handling for situations where the specified files cannot be written due to permission issues or disk space shortage.
4. Validate all file operation outcomes to provide proper function behavior under all circumstances.
5. Confirm the correctness of string replacement values to prevent malformed configurations.
6. Ensure that the usage of this bash function is appropriately documented and communicated to maintain standards of usage and prevent misusage.

3.0.136 `n_ensure_modules_available`

Contained in `lib/node-functions.d/alpine.d/network-module-load.sh`

Function signature: 1d8b435bafefa9c88f65d18b733f3f3374e79fcc326deecf1f196bdf46094d03

3.0.137 Function Overview

The Bash function `n_ensure_modules_available` checks the availability of the kernel modules for the current kernel version, and if the modules are not available, it attempts to make them available. The function starts by looking for the availability of kernel modules directory and the file `modules.dep`. If these are not found, the function tries to find the modloop files in the `/media/*/boot/` directory and mounts it if it has not been mounted. If the modloop files are not found, the function triggers a modprobe to mount them. Lastly, if `modules.dep` is still missing, it generates `modules.dep` using `depmod`.

3.0.138 Technical Description

- Name: `n_ensure_modules_available`
- Description: Checks the availability of kernel modules for the current kernel version. Attempts to make them available if not already present.
- Globals: None.
- Arguments: None.
- Outputs: Logs the progression and result of the availability check and any attempts to make modules available using the `n_remote_log` function.
- Returns: 0 if kernel modules become available. 1 if the function is unable to make the kernel modules available after trying.
- Example Usage:

`n_ensure_modules_available`

3.0.139 Quality and Security Recommendations

1. Validate the kernel version before using it to construct the `modules_dir` path.
2. Handle the errors more efficiently in case of not being able to make modules available.
3. The function could use more comments for self-explanation and ease of understanding to other programmers.
4. The function could use a `trap` mechanism to clean up in case of script failures or script being killed.
5. Make sure the `n_remote_log` function is designed to handle every case without any leakage of sensitive informations.

3.0.140 `n_force_network_started`

Contained in `lib/host-scripts.d/alpine.d/networking-functions.sh`

Function signature: 448c7f84e1898821d351326c82f995cb0da05d48164042795fd8f9ec14ea1b86

3.0.141 Function overview

The function `n_force_network_started()` attempts to manually force the networking service of the system to appear as 'started'. It initially cleans any failed state of the service, then creates necessary state files, and marks the service as 'started'. It verifies the state of the service at the end and returns a boolean value indicating whether the operation was successful.

3.0.142 Technical description

- **Name**: n_force_network_started
- **Description**: This function forces the networking service to appear 'started'. It creates necessary directories, files, and markers, and also double-checks the status of the service.
- **Globals**: None used in this function.
- **Arguments**: No arguments are used in this function.
- **Outputs**: The function outputs log messages indicating the progress of the operation, such as marking the service as started, verifying its status, and error messages if any failures occur.
- **Returns**: The function returns `0` if the networking service is successfully marked as started and verified; returns `1` if it fails to mark the networking service as started.
- **Example usage**: The function can be used like so: `n_force_network_started`. Since it doesn't take any arguments, it's invoked without any.

3.0.143 Quality and security recommendations

1. Although the current function seems to handle errors by logging messages, it could be made more robust by including error exception handling.
2. All filesystem operations are potential points of failure, and should be checked for success/failure.
3. All external commands (such as `mkdir`, `touch`, `sed`, and `rc-status`) should have their return status checked.
4. It may be beneficial to split this function into multiple smaller functions, such as one for directory creation, one for file creation and modification, one for verification. This may increase readability and maintainability of the script.
5. Considering security, it is preferable to avoid the execution of shell commands wherever possible, as they can expose vulnerabilities.
6. There's room for adding more comments explaining the individual steps, making it easier for others to understand the purpose of each command.

3.0.144 `n_force_start_services`

Contained in `lib/host-scripts.d/alpine.d/KVM/install-kvm.sh`

Function signature: 7f43681f2f170c1a4dd9b133c2e8539a06c8e5b262636f6cb4d83f6dc823b233

3.0.145 Function overview

The `n_force_start_services` function is used to ensure the "dbus" and "libvirtd" services are running on a system, starting them directly if necessary rather than relying on the system's init process. The function logs what it's doing for tracking purposes and handles the creation of necessary directories and files for running these services, such as a unique machine-id for dbus and a pid-file for libvirtd.

3.0.146 Technical description

Name: `n_force_start_services`

Description: This function forces the start of the dbus and libvirtd services, bypassing the init process. It checks if these services are already running; if not, it directly starts them and creates required folders and files. If a service fails to start, the function will return an error.

Globals: None

Arguments: None

Outputs: Logs to a remote logging function, `n_remote_log`, about the status of starting services.

Returns: The function returns `0` if all services were started successfully or were already running. It returns `1` if the libvirtd service failed to start.

Example Usage:

```
n_force_start_services
```

3.0.147 Quality and security recommendations

1. Validate the success of directory creation - Currently, the function assumes that `mkdir` operations will always succeed. It may be prudent to confirm their success before proceeding.
2. Use absolute paths - To avoid any mistakes caused by the present working directory, use absolute paths whenever possible.
3. Consider adding error handling for the dbus-uuidgen command as currently its potential failure isn't addressed.
4. Enhanced logging - Depending on the system's logging solution, it might be appropriate to log to more than just a remote log function. Perhaps logging to syslog, capturing error outputs, or raising the severity of tmessages if services fail to start.
5. Generally, it is unsafe to force start of services by bypassing the init system. Instead, one should attempt to resolve the underlying problem that prevents services from starting normally. This function should be treated as a workaround in critical situations, rather than a permanent solution.

3.0.148 `n_get_provisioning_node`

Contained in `lib/host-scripts.d/pre-load.sh`

Function signature: 4e8d2d80a56db706bb5ebb500956281db15dd8b0c1afadab8b2a07d2b711d176

3.0.149 Function Overview

The `n_get_provisioning_node` function is tasked with the responsibility of returning the IP of the default gateway, in other words, the provisioning node. It achieves this functionality using a combination of the `ip route` command to get the IP routing table and the AWK text processing language to parse the output of the command and extract the required field.

3.0.150 Technical Description

- **Name:** n_get_provisioning_node
- **Description:** This function retrieves and returns the default gateway IP in the IP routing table, which is the provisioning node.
- **Globals:** None, this function does not utilize or modify any global variables.
- **Arguments:** This function does not accept any arguments, as it retrieves data from the system directly.

- **Outputs:** The function prints the IP address of the default gateway to stdout.
- **Returns:** The function does not have a return value as such since it directly prints to the stdout. However, if we consider the printed IP as the return, it returns the default gateway IP.
- **Example Usage:**

```
gateway_ip=$(n_get_provisioning_node)
echo "The default gateway (provisioning node) is $gateway_ip"
```

3.0.151 Quality and Security Recommendations

1. To improve quality, the function could handle errors or unexpected outputs from the `ip route` command which may not contain expected default route information.
2. Enhance security by validating and sanitizing the IP before returning it. This could prevent any potential command injection attack if the function output is used elsewhere.
3. Enhance readability and portability of the function by using full command paths, avoiding command shortcuts and by commenting the code more thoroughly.
4. Future versions of this function could consider accepting the interface name as an argument so that it can work in environments with multiple network interfaces.

3.0.152 `n_initialise_opensvc_cluster`

Contained in `lib/node-functions.d/common.d/n_opensvc-management.sh`

Function signature: 01a1305b128f25446f36e9f821d91bb82d4b111a5d1b488cf15e367d620c9125

3.0.153 Function overview

This function, `n_initialise_opensvc_cluster`, is used to initialize a node on an OpenSVC cluster. This function retrieves cluster name, node name and heartbeat type of the cluster from remote resources. It sets the cluster node to the retrieved name and sets the heartbeat type to the retrieved value. It also retrieves and sets node tags. Finally, the function logs the status of the initilisation process throughout the function execution and on completion.

3.0.154 Technical description

- **Name**: `n_initialise_opensvc_cluster`
- **Description**: This function initializes a node on an OpenSVC cluster. It sets the node name and heartbeat type, retrieves and processes the node tags, and logs the process and completion status.
- **Globals**: None.

- **Arguments**: None.
- **Outputs**: Logs status of processes and eventual status of completion.
- **Returns**: Returns 1 in case of failure in setting the heartbeat type or node tags.
- **Example usage**: `n_initialise_opensvc_cluster`

3.0.155 Quality and security recommendations

1. Adding error handling mechanisms or exit conditions could prove beneficial in case of failure in remote variables retrieval. This function assumes that these steps will always succeed, potentially leading to errors.
2. Make sure that tag values are being properly sanitized before their usage in order to prevent potential security issues.
3. Hardcoding the return values is not a best practice. It would be better to define constant variables at the beginning of the script to improve readability and maintainability.
4. It is advisable to include more detailed comments throughout the function to clearly describe what each section of code is doing, especially for complex processes.
5. Consider anonymizing the log data if it contains any sensitive information to ensure data privacy and security.
6. Always verify and validate the values that you are placing in the logs from the remote sources for safety and security.

3.0.156 `n_init_run`

Contained in `node-manager/base/n_init-functions.sh`

Function signature: b41282b6d73e5b45d68377b45c3c0dbfe86780807d723e26cacb5e8d992545a5

3.0.157 Function overview

The function `n_init_run` is a part of a larger bash script that is designed to initiate a sequence of actions, logging progress and any issues. This involves checking an initialization variable, `HPS_INIT_SEQUENCE`, before running any enumerated actions contained within it. For each action, the function checks if it exists and logs an error if it doesn't; if it exists it attempts to execute the action and logs the action's success or failure.

3.0.158 Technical description

- **Name**: `n_init_run`
- **Description**: The function is intended to process an array of actions, `HPS_INIT_SEQUENCE`. For each action, it checks if it's a valid function and

then attempts to execute it. The function has a dependency on another function, `n_remote_log`, which it uses for logging if available.

- **Globals**: `[ HPS_INIT_SEQUENCE: An array of action functions to be executed ]`
- **Arguments**: `[ none ]`
- **Outputs**: Messages detailing the execution of the init sequence, directed to STDERR. If the `n_remote_log` function is available, these will also be delivered as remote logs.
- **Returns**: 0, regardless of the success or failure of the individual actions.
- **Example Usage**:

```
HPS_INIT_SEQUENCE=("action_1" "action_2")
n_init_run
```

3.0.159 Quality and security recommendations

Improvements could be made to the `n_init_run` function to enhance its quality and security.

1. The function is making use of global variables which is not a recommended practice, especially if the scripts are designed to be sourced and used within other scripts. It would be better to make `HPS_INIT_SEQUENCE` an argument to the function.
2. Error handling could be improved in this function. It currently returns a 0 status regardless of whether the actions in the sequence fail or not, which can be problematic down the line if this function is used as a part of larger script and the return status is used to make decisions.
3. The script should make better use of exit codes to reflect the actual outcome of the function. A non-zero exit code should be returned in case of any failures in action execution.
4. The function should probably avoid the use of STDERR for normal operation messages and stick to using it only for errors.

3.0.160 `n_install_apk_packages_from_ips`

Contained in `lib/node-functions.d/alpine.d/alpine-lib-functions.sh`

Function signature: 7de0864578a18b141512aa03e6a8f3e844303e8445cf762fc5657adca810602d

3.0.161 Function overview

This Bash function, `n_install_apk_packages_from_ips`, is specifically designed to automate the process of downloading and installing .apk packages from an IPS (Internet Provider Service). This function takes in an indefinite number of package names as arguments and installs these onto the local system. The process includes the verification

of input package names, determination of the IPS gateway address, accessing the APK repository through the IPS gateway, creation of a temporary directory for operation proceedings, extraction of .apk file names, and identification of the package count.

3.0.162 Technical description

- **Name:** n_install_apk_packages_from_ips
- **Description:** A Bash function meant to automate the process of downloading and installing .apk packages from an Internet Provider Service (IPS).
- **Globals:** No globals are used in the function
- **Arguments:**
  - $1, $2, […]: An array of package names to be downloaded and installed from the IPS. Should be at least one.
- **Outputs:** The function outputs status messages and error messages during its operation, including the IPS gateway, repository and packages, available packages, and failed operations.
- **Returns:**
  - Returns 1 if there are no package names provided, if the IPS gateway address couldn't be determined, if no packages are found in the repository.
  - Returns 2 if the function fails to create a temporary directory or fails to fetch package list from the repository.
- **Example Usage:**

```
n_install_apk_packages_from_ips package1 package2
```

3.0.163 Quality and security recommendations

1. Ensure proper validation of inputs. In this case, check if the package names provided as arguments are valid, non-malicious, and available in the repository.
2. Error handling should be efficient. Make sure all possible error scenarios are captured and addressed.
3. When interacting with an external service, such as fetching data from a URL, always anticipate connectivity issues or unavailability of the service.
4. The temporary directory should be securely deleted after its use to prevent the possibility of any security vulnerabilities.
5. The function should adhere to the principle of least privilege, meaning it should not require or misuse more permissions than necessary to achieve its functionality.
6. Ensure sensitive information, such as an IPS address or repository link, are not exposed or logged inappropriately in your output.

3.0.164 n_install_base_services

Contained in lib/host-scripts.d/alpine.d/BUILD/05-install-utils.sh

Function signature: 4cd6b1fb7ba56a5274df7644a0c4fa2b3d40481f53d79993030a5e76b148313b

3.0.165 Function overview

n_install_base_services is a Bash function designed to install and start base services on a system using the apk package manager. This function starts by defining two local variables (PACKAGES and SERVICES) which hold the names of the packages and services, respectively. The function creates logs of its operations using the n_remote_log function. It updates the apk index, installs the specified packages, and proceeds to enable and start the services one by one. If any of these processes fails, an error log is generated and the function returns 1. If all the steps are successfully executed, a success log is created and the function returns 0.

3.0.166 Technical description

- **name**: n_install_base_services
- **description**: Installs and starts up base services from a predefined list of packages.
- **globals**: None.
- **arguments**: None.
- **outputs**: Logs of the function's processes and potential errors.
- **returns**: 1 if the apk update or package installation fails, 2 if a service fails to start, 0 if the whole process completes successfully.
- **example usage**: n_install_base_services

3.0.167 Quality and security recommendations

1. To improve the flexibility of the function, arguments could be added to allow the user to define specific packages and services rather than working off a predefined list.
2. Robust error handling is already implemented, but detailed error messages that can guide troubleshooting would be beneficial.
3. Incorporate a mechanism to check if the service/package already exists on the system. Thereby, avoiding unnecessary installations or errors.
4. Thoroughly document the function, especially when making changes or updates, to ensure that it remains user-friendly and accessible to other programmers.
5. Always check the validity of packages to be installed to prevent potential security breaches.
6. Be wary of potential injection attacks by sanitizing any user-provided input.

3.0.168 n_installer_cleanup

Contained in node-manager/alpine-3/+INSTALLING/installer-functions.sh

Function signature: c95f46616dfb982eb488533d36ac9d9b32412f1241dd38f6a6aeba176910ca06

3.0.169 Function Overview

The n_installer_cleanup function is primarily used in system operations to undertake cleanup tasks after the installation process. It does this by unmounting filesystems, stopping RAID arrays, wiping filesystem signatures, clearing partition tables (on demand), and resetting system state to allow for re-installation. The function flags for forceful execution and wiping the partition table, providing an extra level of control over how it behaves. This function is an integral part of system recovery and setup scripts.

3.0.170 Technical Description

- **Name:** n_installer_cleanup
- **Description:** The function performs cleanup tasks after an installation, including unmounting filesystems, stopping RAID arrays, wiping filesystem signatures, clearing partition tables on-demand, and resetting system state to allow for re-installation.
- **Globals:** None
- **Arguments:**
  - **$1 (Force flag):** If set to 1, the function executes without asking for user confirmation.
  - **$2 (Wipe partition table flag):** If set to 1, the function clears partition tables.
- **Outputs:** Log messages about the cleanup process. Details of actions to be performed and their results are displayed.
- **Returns:** An exit code to indicate the result of the operation. Returns 1 if no installation devices are found or fails to carry out cleanup. Returns 2 if user cancels operation. Returns 0 upon successful completion of cleanup.
- **Example usage:** n_installer_cleanup --force --wipe-table

3.0.171 Quality and Security Recommendations

1. Place further error checks: Comprehensive error checks can prevent unexpected script behavior if the function encounters unusual circumstances.
2. Consider replacing the echo-based logging system: Current logging using echoes might be replaced with a more robust logging system that can work across multiple scripts.
3. Secure sensitive operations: Certain operations such as wiping partition tables could have significant system consequences. Consider adding additional safeguards against accidental triggers.

4. Handle user input safely: As the script reads user input for confirmation, it might be a good idea to ensure this reading process is done in a secure manner.
5. Confirm erasure is successful: While the function tries to wipe certain filesystems and arrays, it doesn't verify if these actions are successful. It could be worthwhile to check if erasure is indeed successful to prevent potential data remnants.

3.0.172 `n_installer_detect_target_disks`

Contained in `node-manager/alpine-3/+INSTALLING/installer-functions.sh`

Function signature: 6968bad72645a951cd649277eb09aee62c1718628ae0c39ceefd1a11567253dc

3.0.173 Function overview

The function `n_installer_detect_target_disks()` is designed to scan and select disks suitable for an installation of an Operating System. During its execution, this function checks if a RAID configuration is requested. If that condition is fulfilled, it searches for two suitable disks. Otherwise, it carries on in single-disk mode. To qualify as suitable, a disk must be non-removable, of the right device type, a whole disk (not a partition), and has its size greater than or equal to a specified minimum. If the function finds suitable disks, it checks whether they are clean, i.e., do not have existing data that might interfere with the new installation. This function also handles error conditions, such as when no suitable disks are found or the disks are not clean.

3.0.174 Technical description

- **name**: `n_installer_detect_target_disks`
- **description**: Searches for and selects disks that are suitable for OS installation. Supports RAID configuration and single-disk mode.
- **globals**:
  - `ROOT_RAID`: determines whether a RAID configuration is requested (`1`) or not (`""`, `!=1`)
- **arguments**: None
- **outputs**: Logs messages about the progress and errors encountered during the execution.
- **returns**:
  - `0`- if at least one disk is found and all found disks are clean
  - `1` - if no suitable disks are found or failed to store the OS disk to IPS
  - `2` - if less than 2 disks are found while `ROOT_RAID` is set to `1`
  - `3` - if installation cannot proceed due to unclean disks
- **example usage**: `n_installer_detect_target_disks`

3.0.175 Quality and security recommendations

1. Implement stricter error checks and return different codes for distinct types of errors. This would help with debugging and allow better reaction to specific errors.
2. Provide a more secure way to clean disks. Overwriting existing data with zeros could make it harder for malicious actors to recover the old data.
3. Make sure that user-specified parameters (like `ROOT_RAID` value) are sanitized properly. Incorrect values should either produce an intelligible warning or be defaulted to safe values to prevent unexpected behavior.
4. Include input validations for making the function robust, especially when verifying the disk type.
5. Always ensure that the system being worked on has proper permissions to avoid any security breaches.

3.0.176 `n_installer_finalize`

Contained in `node-manager/alpine-3/+INSTALLING/installer-functions.sh`

Function signature: 392e9e5a731910ef6b5ef1c40d329527a6b672723f46386d932ca6ecca9f7625

3.0.177 Function overview

The `n_installer_finalize()` function is primarily utilized to finalize the installation process of a Unix system. After giving an info log stating the initialization of the finalization process, the function syncs filesystems and checks if RAID1 was used by verifying if specific md devices exist. If these devices are present, it saves the mdadm configuration, generates mdadm.conf, and logs the config. The function also checks if mdadm is installed and enabled in the target. If not, it installs mdadm and enables it. Afterwards, it updates the initramfs to include mdadm. The function then syncs again and unmounts the filesystems. Lastly, it updates the host_config state to INSTALLED and reboots the system.

3.0.178 Technical description

Name: `n_installer_finalize()`
Description: Finalizes the Unix installation process, ensures filesystems sync, checks for RAID1 and updates the related configuration, unmounts filesystems, updates host_config state to INSTALLED, and reboots the system.
Globals: None
Arguments: None
Outputs: Log outputs providing information/debug/error info about the function status.
Returns: Exit status 0 if everything runs successfully, 1 if there is a failure unmounting filesystems, 2 if there is a failure updating the state to INSTALLED.
Example Usage: `n_installer_finalize`

3.0.179 Quality and security recommendations

1. Consider adding more validation for various steps in the installation process. For example, after each critical step, it would be best to ensure the step was successful before proceeding.
2. Catching and handling more specific exceptions could help identify issues and resolve them more efficiently.
3. Utilize a hash function to protect sensitive configuration information being logged.
4. Apply the least privilege principle by limiting the permissions of the service/user executing this function.
5. Conduct regular security audits on the function to ensure ongoing security. Maintain up-to-date documentation and ensure adherence to the latest security standards and best practices.

3.0.180 `n_installer_format_partitions`

Contained in `node-manager/alpine-3/+INSTALLING/installer-functions.sh`

Function signature: 0b4a0dbbfa19524cd835648932858b139e035c5ca2a01459ac2743b4c74bb981

3.0.181 Function overview

The `n_installer_format_partitions` is a Bash function designed to format and mount boot and root partitions. This function reads device paths from a host config, checks for the necessary commands, and installs them if they are not already present. It then formats the boot and root partitions, mounts them on the filesystem, and generates a file system table (fstab). The function also includes extensive logging for debugging and troubleshooting purposes.

3.0.182 Technical description

- Name: `n_installer_format_partitions`
- Description: This function formats and mounts the boot and root partitions on a device while ensuring all the necessary utilities are available and logs the process throughout.
- Globals: None
- Arguments: None
- Outputs: Logs the progress of operations, including errors, to the standard output or error output.
- Returns: Returns 0 on success, and 1, 2, 3, or 4 on various errors.
- Example Usage:

```
n_installer_format_partitions
```

3.0.183 Quality and security recommendations

1. Error handling is well implemented in this function, however, more specific error messages could be beneficial in debugging.
2. In the event of an error, the function attempts to unmount the /mnt directory. However, for the /mnt/boot directory, it only attempts to unmount it once after failure. It might be beneficial to attempt to unmount /mnt/boot in other error situations as well.
3. When fetching UUIDs for boot and root devices, there's no check for successful command execution. It might be useful to return an error if fetching UUID fails.
4. The `umount` calls are fire-and-forget. Although unmounting often fails if a device is busy, in this case, failing to unmount might mean that important filesystem changes don't get written, and it would be safer to check the exit status.
5. Lastly, when creating the swap file, the code assumes that the dd command will create a 1GB file. However, if disk space is insufficient, a smaller file, or no file, will be created. A subsequent check for the swap file's size will increase safety.

3.0.184 `n_installer_install_alpine`

Contained in `node-manager/alpine-3/+INSTALLING/installer-functions.sh`

Function signature: db43e0af9d850dcdf19761f9dc9c6407c3a7918276fbcadf8609fa397c5ba31f

3.0.185 Function overview

The `n_installer_install_alpine` function is created for installing the Alpine OS on a given mount point in a Linux system. It checks the mount points, retrieves required variables from the host configurations such as os_id and repo_path, and constructs repository URLs. It then configures apk repositories, updates apk index, and checks if required tools, such as setup-disk and grub-install, are available in the environment. Subsequently, the function makes use of setup-disk to install Alpine base system, installs the GRUB bootloader, and verifies the installation.

3.0.186 Technical description

- **name**: `n_installer_install_alpine()`
- **description**: Installs Alpine OS at a specific mount point in a Linux system.
- **globals**: None
- **arguments**: None
- **outputs**: Various informational, debug, error and warning messages are logged using the `n_remote_log` command to indicate progress or issues during execution of the function.
- **returns**:

  - 0: Successful installation
  - 1: Error while retrieving os_id or repo_path or determining IPS host
  - 2: /mnt or /mnt/boot is not mounted
  - 3: Error while creating or updating /etc/apk/repositories file or updating apk index
  - 4: Error while installing Alpine base system or GRUB bootloader or verifying the installation

- **example usage**:

```
n_installer_install_alpine
```

3.0.187 Quality and security recommendations

1. Make sure that the /mnt and /mnt/boot directories are correctly mounted in order to avoid any mount point errors.
2. Proper error handling should be in place for scenarios where necessary commands might not be found.
3. Ensure the `os_id` and `repo_path` are correctly set in the system to avoid any errors.
4. For security reasons, minimal permissions should be granted that are required to execute the function.
5. Do validate the function success or failure, using the return code, wherever the function is used.
6. Explicitly validate and sanitize all derived data as a security measure.

3.0.188 `n_installer_install_hps_init`

Contained in `node-manager/alpine-3/+INSTALLING/installer-functions.sh`

Function signature: 9960fe4281769e59b004547d9e6d97cb9ccaf9d6e26457a12cda79bf2f57378f

3.0.189 Function overview

The function `n_installer_install_hps_init` is responsible for setting up the initial conditions for a node server to operate within a target HPS (Hyperscale Processing System) environment. Among its settings, this function configures the persistent network, the node functionalities, the hostname of the system, installs required packages (like bash), enables SSH service, and allows root login without a password.

3.0.190 Technical description

- **name**: `n_installer_install_hps_init`
- **description**: This function installs and configures a new HPS environment on the target system, enables key services, and allows root login with no password in a development mode.

- **globals**: No global values are directly changed by this function.
- **arguments**: The function does not explicitly use arguments in its implementation.
- **outputs**: Outputs are handled by `n_remote_log`, sending messages regarding operation statuses to a log system.
- **returns**: It returns 0 if successful operations, and 1 or 2 for errors.
- **example usage**: `n_installer_install_hps_init`

3.0.191 Quality and security recommendations

1. The function currently allows root login without a password for SSH, which can pose severe security risks. Therefore, it is strongly recommended to disable the `PermitRootLogin yes` and `PermitEmptyPasswords yes` SSH settings for environments outside of development.
2. Ideally, there should be error handling or exception handling for critical operations such as creating directories, copying files, etc.
3. The function could benefit from more granular and functional separation, enhancing testability and readability.
4. Consider using secure and unique log paths to prevent potential log forgery.
5. Although the script has many comments, it would be beneficial to have more explicit and clearer documentation about the expected environment and preconditions.

3.0.192 `n_installer_load_ext4_modules`

Contained in `node-manager/alpine-3/+INSTALLING/installer-functions.sh`

Function signature: 7e5f7a3859012bb1d363fe9ffd312363ec41451d44c34eb97eb444cc591cd7d4

3.0.193 Function overview

This function, `n_installer_load_ext4_modules()`, is designed to verify if the ext4 filesystem is available, and if not, it attempts to load the necessary modules for ext4 to become available. If the function encounters problems during verification or while loading modules, it logs error messages and returns a non-zero exit status.

3.0.194 Technical description

Name: n_installer_load_ext4_modules Description: The function checks if ext4 filesystem is available. If not, it attempts to load the necessary modules for ext4. If the function encounters any issues during verification or during loading of the modules, it would log error messages and returns a non zero exit status.

Globals: None

Arguments: None

Outputs: Log messages indicating the state of ext4 filesystem and the modules' loading process.

Returns: 0 if ext4 is already available or the modules are successfully loaded. 1 if ext4 is not available after attempting to load the modules, or if any other error occurred during the process.

Example usage:

```
source n_installer_load_ext4_modules.sh
n_installer_load_ext4_modules
```

3.0.195 Quality and security recommendations

1. Validate inputs: Though the function doesn't take any arguments, it relies on a global environment which must be prepared properly. Validations must be done to ensure all dependencies are correctly prepared.
2. Error handling: The function does a good job in error handling, ensuring that if any step fails, an appropriate status code is returned along with error logging. Continuation of the processing is stopped as soon as an error occurs. The same practice should be kept in other similar scripts.
3. Security: Ensure that the script is executed with the least privilege required to mitigate risks associated with uncontrolled scripts execution.
4. Maintainability: Consider adding further comments in the code to improve readability and ease future modifications.
5. Logging: Make sure logs do not contain sensitive information, as they may be exposed in plain text logs. Consider ways to enable/disable debug logging depending on the environment the script is being run in.

3.0.196 `n_installer_partition_disks`

Contained in `node-manager/alpine-3/+INSTALLING/installer-functions.sh`

Function signature: 481842e1c9ac4d126f0266c400ecbb23ec6be762d9b3a2c2d704968f0eb99955

3.0.197 Function overview

The `n_installer_partition_disks()` function is responsible for automating the disk partitioning process on a host machine. It takes a list of disks from the host configuration and partitions them in different modes based on the number of disks (Single disk mode for one disk, RAID1 mode for two disks). The function also checks and installs required tools if they aren't already installed and handles special naming convention for NVMe disks.

3.0.198 Technical description

- **name**: `n_installer_partition_disks`
- **description**: This function automates the disk partitioning process based on the configuration on host machine.
- **globals**: [`IFS`: Internal field separator used for splitting a string into array, `os_disk`: A variable fetched from the host configuration containing the disk(s) to be partitioned]
- **arguments**: No arguments required.
- **outputs**: Log messages indicating the steps and proceedings or any errors encountered.
- **returns**: `1` if errors like failed to read `os_disk` from host config or empty `os_disk` or invalid disk count; `2` if fails to install required tools or partitioning failed.
- **example usage**: `n_installer_partition_disks`

3.0.199 Quality and security recommendations

1. Error message should not disclose too much information about what went wrong for security concerns.
2. The function can be made more secure by performing greater validation of the `os_disk` value before attempting disk operations.
3. Consider handling cases where disk count is more than 2.
4. For enhancing this function's maintainability, modularization can be done by breaking down larger chunks of code into smaller functions.
5. It is recommended to have a way to roll back changes if any step in the function fails.
6. Consider handling the failure conditions (like failing to install required tools or partitioning) more robustly, possibly with retries or alternatives.

3.0.200 `n_installer_run`

Contained in `node-manager/alpine-3/+INSTALLING/installer-functions.sh`

Function signature: d059d8ab8cb8ce3656dca63dc8b16af5dfa85629dbb5732522194909b33d1f04

3.0.201 Function overview

The `n_installer_run` is a Bash function designed to automate the process of installing the Alpine SCH system on a remote host. It takes no arguments and expects to run in an environment where other `n_` functions and global variables are predefined. It sequentially calls other installation functions, providing detailed logging of the execution process, and updates the status of installation on the remote host.

3.0.202 Technical description

- **Name:** `n_installer_run`
- **Description:** The function sequentially runs system installation steps, logs the actions and errors, and keeps the remote host updated with the state of the installation.
- **Globals:**
  - `STATE`: reflects the current state of the installation;
  - `INSTALL_ERROR`: reflects the error during installation steps.
- **Arguments:** None
- **Outputs:** Logs messages to the remote log about the status of each installation step.
- **Returns:**
  - `1` if Disk detection step failed;
  - `2` if Partitioning step failed;
  - `3` if Formatting step failed;
  - `4` if Alpine installation step failed;
  - `5` if HPS init installation step failed;
  - `6` if Finalization step failed;
  - `0` in event of reaching the bottom of function, implying an error since finalization is supposed to reboot.
- **Example usage:** The function is called without arguments, like so: `n_installer_run`

3.0.203 Quality and security recommendations

1. Create local worker functions or scripts for each installation step to improve readability, testability, and maintainability of the code.
2. Use more descriptive identifiers for the global variables to avoid potential naming conflicts.
3. Handle and validate possible errors and exceptions in a more comprehensive way, focusing on potential system issues.
4. Include more detailed logging for troubleshooting purposes and clarity.
5. Verify permissions and authenticate all system-level actions to improve security.
6. Implement timeouts to prevent hanging processes during the installation steps.

3.0.204 `n_installer_verify_disk_clean`

Contained in `node-manager/alpine-3/+INSTALLING/installer-functions.sh`

Function signature: 5e224012f195010efc59a14eba77a63742ab7ea705758887f95d5c7cb37fd39b

3.0.205 Function Overview

The function `n_installer_verify_disk_clean` is a shell function designed to verify if a given disk is clean (i.e., no partitions, file systems, or other structures such as RAID superblocks, LVM physical volumes, or ZFS labels exist on it). The function first defines local variables for the given disk and buffers to hold whether the disk is clean or not and any issues found. It then sequentially verifies for cleanliness by checking for various data structures. If any of the checks fail, the disk is marked as not clean, and an issue is logged. Finally, if the disk is not clean, a warning is logged and returned.

3.0.206 Technical Description

- **Name:** `n_installer_verify_disk_clean`
- **Description:** Verifies if a given disk is clean of any data structures.
- **Globals:** No Global variables required.
- **Arguments:**
  - `$1: disk` - The disk to verify.
- **Outputs:** Logs any issues found on the disk or logs if the disk is clean.
- **Returns:**
  - `0` if the disk is clean.
  - `1` if the disk is not clean.
- **Example usage:** `n_installer_verify_disk_clean /dev/sda`

3.0.207 Quality and Security Recommendations

1. The function uses several external commands and depends on their availability on the system. It is recommended to handle the cases where these commands are not available more gracefully to avoid breaking the script.
2. It's recommended to have additional error handling around the external commands to capture and handle the error conditions effectively. For instance, a common bash idiom is to use the `set -e` option to halt the script if any command returns a non-zero status.
3. There could be security concerns if untrusted input can affect the name of the disk being verified. Always ensure that the input to this function is sanitized and validated.
4. Consider adding more comments in the function to make the purpose and functionality of the code more readable and understandable.

3.0.208 `n_install_kvm`

Contained in `lib/node-functions.d/alpine.d/KVM/install-kvm.sh`

Function signature: 145553d28c54a7d2f2d58c019aae336602f2abc5947c73a67d2940901ca02f6b

3.0.209 Function Overview

This bash function, `n_install_kvm()`, serves to install and set up virtualization packages for a KVM (Kernel-based Virtual Machine) setup. This includes installing various packages like qemu-system-x86_64, qemu-img, libvirt, libvirt-daemon, and dbus. Then it starts the dbus service and sets libvirtd to start at boot-time, checks network functionality, and depending on boot sequence, starts the libvirtd service. The function is verbose, logging steps and changes as it proceeds, and updates the status of 'virtualization_status' and 'virtualization_type' variables with the help of `n_remote_host_variable`.

3.0.210 Technical Description

- **Name:** `n_install_kvm()`
- **Description:** This function is designed with focus on installing and setting up virtualization packages on a system. It is highly informative, as it logs each step or stage of the process, and is built to handle failures gracefully.
- **Globals:** None
- **Arguments:** This function does not take any arguments.
- **Outputs:** Logs of the steps taken, status of services and any error or warning messages.
- **Returns:** `0` if the installation is successful, `1` if the virtualization packages installation fails, and `2` if the startup of libvirtd service fails.
- **Example Usage:** `n_install_kvm`

3.0.211 Quality and Security Recommendations

1. Use more specific exit codes for each type of failure to make error handling more consistent.
2. Enhance error messages to include more specific information about what went wrong.
3. Avoid using pipes when unnecessary to prevent unnecessary subshells.
4. Use absolute paths for binaries to avoid potential issues with malicious code injections.
5. Consider setting read-only variables for critical data to prevent unintentional modification.
6. Check service status or existance before trying to start them to avoid unnecessary executions.
7. Make sure that all error messages are logged to a centralized diagnostic location. This will make recovering from failures easier.
8. Consider using a package manager that supports transactional operations to ensure all packages are installed successfully or not at all in case of any error.

3.0.212 n_install_opensvc_packages

Contained in node-manager/base/n_install_opensvc_packages.sh

Function signature: 4f783be1e0de1e418348cd98de433e9ccea079845d27f4d08d05d4732d139e65

3.0.213 Function Overview

The Bash function n_install_opensvc_packages() is designed to support the installation of OpenSVC packages in an operating system-neutral manner. It initially logs a message indicating the start of OpenSVC package installation. Subsequently, it detects the operating system in use, and based on that, it decides the suitable package manager to use. If the OS is Alpine Linux, it selects the APK package manager, whereas for Rocky/RHEL, rpm/dnf is chosen. For Rocky/RHEL, there's a note indicating that its package installation is yet to be implemented. When the OS isn't recognized, it logs an error message and stops execution by returning 1.

3.0.214 Technical Description

- Name: n_install_opensvc_packages

- Description: Install OpenSVC packages through a suitable package manager based on the detected operating system.

- Globals: [n_remote_log: Logs messages for remote operations]

- Arguments: [None]

- Outputs: Log messages indicating the start of installation, errors such as unrecognized OS, and unimplemented sections for specific OS are outputted.

- Returns: 0 if the operation was successful, 1 if the operation failed.

- Example usage:

```
n_install_opensvc_packages
```

3.0.215 Quality and Security Recommendations

1. Verify the package source before their installation to ensure that there's no malicious content involved.
2. It is encouraged to put package names in a separate read-only file or similar, to prevent tampering and accidental modifications.

3. The TODO section regarding the implementation of package installation for Rocky must be completed.
4. Currently, the function doesn't handle gracefully in the case of an unknown OS. Instead, you should attempt to identify the most common OS's in your operational footprint and add support for them.
5. Absence of input validations poses a security risk, consequently it's recommended to incorporate checks for input validation to prevent injection attacks.

3.0.216 `n_interface_add_ip`

Contained in `lib/node-functions.d/common.d/n_network-functions.sh`

Function signature: 6a786746bbd31691fcadd616683c9d48446f70621c619cbce04fff61c3e311ed

3.0.217 Function overview

This function, `n_interface_add_ip`, is used to add an IP address to a specified network interface. This is a low-level operation typically used in network setup or reconfiguration scripts. The function takes three arguments: the name of the network interface, the IP address to be added, and the netmask. The final argument, the netmask is also converted to CIDR notation as needed. If provided IP address is already assigned to the network interface, the function logs this condition and returns without making changes.

3.0.218 Technical description

- **name**: `n_interface_add_ip`
- **description**: Adds an IP address to a specified network interface. The function also converts the provided netmask to CIDR form.
- **globals**: None
- **arguments**:
  - `$1`: The network interface to which the IP address will be added.
  - `$2`: The IP address to add.
  - `$3`: The network mask associated with the IP address.
- **outputs**: Notifies user if the provided IP already exists on the specified interface.
- **returns**:
  - 1 if invalid arguments are passed.
  - 0 if the operation is successful or the IP address already exists on the interface.
- **example usage**: `n_interface_add_ip eth0 192.168.1.2 255.255.255.0`

3.0.219 Quality and security recommendations

1. Ensure to use this function with valid and trusted input only. The function does not sanitize or validate input, therefore, untrusted input could potentially lead to issues.
2. Return codes should be handled or verified after calling this function to ensure the operation was successful.
3. Implement error handling to account for potential failure in IP assignment.
4. The function can potentially accept a CIDR notation netmask as it doesn't perform validation. This could be a useful extension or a potential issue needing to be addressed, depending on your context. If you need to restrict addresses to a traditional 4 octet netmask, you could add a check for this.
5. Avoid using this function in scripts that don't need to alter network configurations. Using such functions can be a potential security risk.

3.0.220 `n_interface_unconfigure`

Contained in `lib/node-functions.d/common.d/n_network-functions.sh`

Function signature: 200deef283b562d77f68a27e48ff4ae382975c0cd282ed282f1796d237824bcb

3.0.221 Function Overview

The function `n_interface_unconfigure()` is designed for network interface management in a bash shell. It accepts three parameters: interface name (mandatory), action to perform (optional, default is 'flush') and a force flag (also optional). The function does different actions based on the parameter. It may flush IPs, delete a VLAN interface, or bring down the interface, giving logs and diagnostic messages along the way. If used improperly, the function provides usage tips.

3.0.222 Technical Description

- **Name:** `n_interface_unconfigure`
- **Description:** This function manages network interfaces, executing certain actions depending on the given parameters.
- **Globals:** None
- **Arguments:**
  - $1: `interface` - the name of the interface on which the action will be performed
  - $2: `action` - a specific action for the function to execute (either 'flush', 'delete' or 'down')
  - $3: `force` - a flag that, when set to 'force', will force the execution of certain actions despite warnings.

- **Outputs:** Logs and diagnostic messages about the actions being taken, warnings in case of potential risky actions, errors when the desired actions can't be performed.
- **Returns:** 1 in case of errors and 0 if the function finishes normally
- **Example usage:**
  - To flush IPs from an interface: `n_interface_unconfigure eth0 flush`
  - To forcefully flush IPs from an DHCP-assined interface: `n_interface_unconfigure eth0 flush force`

3.0.223 Quality and Security Recommendations

1. Validate parameters further: Currently, the function has limited parameter validation. It might be beneficial to validate the format and correctness of the `interface` argument.
2. Implement error handling for all critical operations: Currently, some operations like `ip addr flush dev "$iface"` are executed without checking their success or failure.
3. Use descriptive and consistent error messages: This could improve debuggability.
4. Always quote your variables: This can prevent unwanted globbing and word splitting.
5. Be cautious when forcing actions: The 'force' parameter overrides some safety measures. Always ensure it's absolutely needed before using.

3.0.224 `n_ips_command`

Contained in `lib/node-functions.d/pre-load.sh`

Function signature: 49296795b09096589391a79f5e195603621ed9b9a086011ba67bdad22202778c

3.0.225 Function overview

The `n_ips_command` function is a Bash script which sends requests to an IPS gateway (a specific type of network device). It builds a query string from the input command and an optional list of parameters, sends a POST request to the gateway with this information, then checks the HTTP status code in the response. If the request was successful the function will output the response to standard out (stdout). If any problems are detected, an error message is stored in the `N_IPS_COMMAND_LAST_ERROR` variable for later use.

3.0.226 Technical description

name n_ips_command - Sends a command to the IPS gateway, by preparing and sending a POST request.

description Parses the command and additional parameters passed to the function. It builds a query string, encodes any URL specific characters and sends to the node provisioned by the IPS gateway. The response from the POST request is evaluated for error handling. If response is successful, it prints the response output.

globals

- `N_IPS_COMMAND_LAST_ERROR` : This is used to store the most recent error that has occurred.
- `N_IPS_COMMAND_LAST_RESPONSE` : This is used to store the response of the most recent executed command.

arguments

- `$1` : Command to be executed on the IPS gateway
- `$@` : Additional parameters for the command in the format param=value (optional)

outputs

- Prints the response received from executing the command on the IPS gateway to standard out

returns

- `1` : If fails to determine the IPS gateway
- `2` : If curl exits with a non-zero status code
- `3` : If the request ends with any HTTP Error code starting with 4 or 5
- `0` : If all operations are successful

example usage `n_ips_command YOUR_COMMAND Param1=Value1 Param2=Value2`

3.0.227 Quality and security recommendations

1. Avoid storing sensitive data in global variables. If needed, clear or overwrite them as soon as their purpose is served.
2. As curl's −w write out option is used which enables cURL to output debug information, ensure no sensitive data is being outputted for security reasons.
3. URL encoding is not applied on the key passed as argument which as of now is not a problem but might become a security risk if these keys are not standardized, and can contain special characters.
4. Add input validation techniques to ensure sensible and safe values are passed as arguments.

5. Handle other potential HTTP status codes, such as 3xx redirect responses, to make your script more robust.

3.0.228 `n_keysafe_request_token`

Contained in `lib/host-scripts.d/common.d/keysafe-node.sh`

Function signature: 43c0b2dc4a1a9560d4393853037cc9eaf34c1e671a67475620b7086e407cea75

3.0.229 Function overview

The `n_keysafe_request_token` function is designed to request a token from the IPS via the `n_ips_command` wrapper, given a specific purpose. This function first validates the provided purpose argument and if valid, the token request is initiated. The function also handles failures in communication with the IPS and checks the response for any errors. If no errors are found, the function filters out warning messages and extra output, to finally return the token.

3.0.230 Technical description

- Name: `n_keysafe_request_token`
- Description: This function is responsible for requesting a token from IPS. It validates the provided purpose, ensures communication with IPS, and checks IPS's response for errors. Upon successful request, it cleans up the response by filtering out any warnings or extra output and returns the token.
- Globals: None
- Arguments:
  - `$1: purpose` - description of the purpose for which the token is requested
- Outputs:
  - On success, returns the token requested from IPS.
  - On failure, outputs error messages indicating the lack of required purpose, failure in communication with IPS, or an error contained in the response.
- Returns:
  - `0` if the token is successfully requested and returned
  - `1` if the communication with IPS failed
  - `2` if the required argument purpose is not provided
  - `3` if the response from IPS contains an error
- Example usage:
  - `n_keysafe_request_token "password_reset"`

3.0.231 Quality and security recommendations

1. Implement stricter input validation. The current version accepts any non-empty string as a valid purpose.

2. Replace the `echo` command with `printf` to avoid potential issues with input that could be treated as options or flags.
3. Consider encrypting the token when sending and decrypting once received to ensure its security while in transit.
4. Keep track of the number of token requests and limit them to prevent potential abuse of the function.
5. Include more granular error checking, such as checking for specific types of errors returned in the response.
6. Include thorough logging for any errors or exceptions for monitoring and debugging. Ensure sensitive information isn't logged in such process.

3.0.232 `n_load_kernel_module`

Contained in `lib/host-scripts.d/alpine.d/lib-functions.sh`

Function signature: e8d3e185dcbaf9a04014e264ef088ba7b3adf4b74b818ea01d0f4f6ddc3c93d9

3.0.233 Function Overview

The `n_load_kernel_module` function is designed to handle the loading of a specified module within the Linux operating system kernel. The primary function is to check whether a specified module is already loaded, and, if necessary, load it into the system. Errors are managed and reported on an ongoing basis, allowing the script to respond to various issues such as unmounted modloops and undetectable kernel versions.

3.0.234 Technical Description

- **Name:** `n_load_kernel_module`
- **Description:** This function attempts to load a specified Linux kernel module by verifying if it's already loaded, confirming modloop is mounted, detecting the kernel version, finding the module file, and lastly, loading the module.
- **Globals:** None
- **Arguments:**
  - `$1: module_name` - The name of a Linux kernel module that will be attempted to load.
- **Outputs:** Log messages about process stages and any errors encountered.
- **Returns:**
  - `0` if the module is successfully loaded or is already loaded.
  - `1` if the modloop isn't mounted or the module directory is not found.
  - `2` if the module itself isn't found.
  - `3` if there is failure in loading the module.
- **Example Usage:** `n_load_kernel_module driver_name`

3.0.235 Quality and Security Recommendations

1. Implement more robust error checking and handling mechanisms for the edge cases not currently managed by the function.
2. Avoid directly echoing errors to STDERR. Instead, make use of dedicated logging functions or systems to register error logs.
3. Develop code to verify kernel module names, this will make the component more robust against malicious input or any textual mistakes.
4. Avoid using commands that could potentially expose the system to command injection attacks.
5. If the function will run with administrative or superuser permissions, extra caution should be taken to ensure that input is properly sanitized.

3.0.236 `n_load_remote_host_config`

Contained in `lib/host-scripts.d/pre-load.sh`

Function signature: deea1cfd1cac05c37febb11a35389234fe97871c7dce4258577047d2511a289a

3.0.237 Function overview

The `n_load_remote_host_config` function is used to load the configuration of a remote host. First, it executes the `host_get_config` command of the `n_ips_command` function and stores the output in the variable `conf`. In case this operation fails, it reports the problem and terminates the operation with a return status of `1`. If the command is executed successfully, it proceeds to execute the contents of `conf`, essentially loading the remote host's configuration.

3.0.238 Technical description

- **Name:** `n_load_remote_host_config`
- **Description:** This bash function loads and executes the configuration of a remote host.
- **Globals:** [conf: Stores the remote host configuration obtained from the `n_ips_command` function]
- **Arguments:** None.
- **Outputs:** Logs indicating failure or success in loading the remote configuration.
- **Returns:** `1` if unable to load the host configuration; no explicit return if loading is successful.
- **Example usage:** `n_load_remote_host_config`

3.0.239 Quality and security recommendations

1. Do an input validation check for `n_ips_command` to ensure only intended commands are executed.

2. Add better error handling for both expected and unexpected errors.
3. Provide a clear, user-friendly message for logging instead of just "Failed to load host config."
4. Avoid using `eval` due to security reasons as it allows command injection. If it's necessary to use `eval`, ensure the content of `conf` is strictly checked and sanitized.
5. Implement logging for the function's successful operation completion.

3.0.240 n_network_find_best_interface

Contained in `node-manager/base/n_network-functions.sh`

Function signature: 6460d83e233fbb1d6aee3d49a4c4ee8e3c1c2b0a9e11a765b0eaf94aaad8b4a9

3.0.241 Function overview

The function `n_network_find_best_interface()` is used to find the fastest network interface available that meets a certain minimum speed criteria and a network state criteria. It reads the interface descriptors, including MAC address, MTU, speed, and the driver from the `n_network_get_interfaces` function. These are used to determine the speed and state of each interface. The function then checks the speed of each interface and compares it with the minimum required speed. If the speed of an interface meets the minimum requirement, the function compares this speed with the speed of other interfaces to define the fastest one. The function returns the fastest interface; if no interface meets the requirements, it returns an error.

3.0.242 Technical description

- **Name**: `n_network_find_best_interface`
- **Description**: This function finds the fastest network interface that matches the given state and minimum speed requirements.
- **Globals**: None
- **Arguments**:
  - `$1`: minimum required speed for the interface (default: 0)
  - `$2`: the required state for the interface (default: 'up')
- **Outputs**:
  - Prints the name of the fastest qualifying interface.
- **Returns**:
  - 0 if a qualifying interface is found.
  - 1 if no qualifying interface is found.

- **Example usage**: The following call finds the fastest network interface that is currently up and has a minimum speed of 100.

  ```
  n_network_find_best_interface 100 'up'
  ```

3.0.243 Quality and Security Recommendations

1. Check the reliability of the input source (`n_network_get_interfaces`), as it directly relates to the output of this function.
2. Avoid parsing command line arguments directly, as they might contain dangerous code.
3. Look for ways to refactor nested conditionals for improved readability.
4. Consider handling unexpected cases, for example, non-numeric inputs for speed.
5. Always verify and sanitize the inputs before using them.
6. Consistently comment code to ensure clarity for others and for future reference.
7. Use shellcheck or other linters to find common shell script problems.

3.0.244 `n_network_find_interface`

Contained in `lib/node-functions.d/common.d/n_network-functions.sh`

Function signature: 33c65883d129b7ec6a881e837330768247f028034c9d53eefc55078d85c68721

3.0.245 Function overview

The function `n_network_find_interface` is utilized to find and return the name of a network interface. The specific state ('up' or 'down') of the interface can be provided as an argument. If no argument is provided, the function will return any interface that is found. In the process, it skips non-physical interfaces for coherence and better performance.

3.0.246 Technical description

- **Name**: n_network_find_interface
- **Description**: This function is responsible for finding and returning the name of a network interface based on its state. The state can vary as 'up' or 'down'. It excludes non-physical interfaces.
- **Globals**: None
- **Arguments**:
  - $1: The preferred state of the interface. It can take the values 'up' or 'down'. If no value is provided, it will default to 'any'.
- **Outputs**: Name of the network interface found.
- **Returns**:

  - 0: If a network interface matching the preferred state is found.
  - 1: If no network interface matching the preferred state is found.

- **Example Usage**:

```
n_network_find_interface 'up'
```

3.0.247 Quality and security recommendations

1. It's advisable to handle errors that might occur when reading the state of a network interface. This can be done by redirecting STDERR to a log file or to /dev/null.
2. The function could offer the flexibility to search for a network interface based on additional parameters other than the state.
3. Security could be improved by validating the input argument to ensure that it's either 'up', 'down', or 'any'.
4. Better comments within the function could improve its readability and maintainability. This function could be part of a larger codebase and a good commenting practice would help other developers understand it quickly.

3.0.248 `n_network_find_unconfigured_interface`

Contained in `lib/node-functions.d/common.d/n_network-functions.sh`

Function signature: 4b8cced8949437d8332391a0d4ab87a48036672dee0cfc1e783b64bae29baacd

3.0.249 Function overview

The `n_network_find_unconfigured_interface` is a Bash function that searches all available interfaces and identifies an "unconfigured" one. This selection is based on two primary criteria: the number of assigned IP addresses (the fewer the better) and the connection speed (the faster the better). The function counts the assigned IP addresses for each interface, and if an interface is found with no IP addresses and a decent speed, it is immediately chosen. If not, all interfaces are scrutinized, and the one with fewest IP addresses and highest speed is chosen.

3.0.250 Technical description

- Name: `n_network_find_unconfigured_interface`
- Description: A Bash function to find an unconfigured network interface based on the number of their assigned IPs and their connection speed
- Globals: N/A
- Arguments: N/A
- Outputs: Outputs the name of the selected unconfigured interface. If no suitable interface can be found, no output is printed.
- Returns: Returns 0 if an interface is found, otherwise no return value.

- Example Usage:

```
$ n_network_find_unconfigured_interface
```

3.0.251 Quality and security recommendations

1. Add error handling: The function doesn't currently handle any exceptions, thus, adding error handling for unforeseen situations could enhance its reliability.
2. Check input validation: Even though this function doesn't accept any arguments, it might be beneficial to check the validity of the information retrieved from interfaces.
3. Avoid command injection: Make sure that the values assigned to the variables within function are not user-supplied inputs to prevent command injection attacks.
4. Limit permissions: Run the function with the least privileges required to reduce the impact of potential security vulnerabilities.
5. Logging: Consider adding logging functionality to help with debugging and monitoring.
6. Code comments: Improve the comments for code readability and maintainability.
7. Testing: The function should be extensively tested to ensure that it works as expected in different environments and edge cases.

3.0.252 `n_network_get_interfaces`

Contained in `lib/node-functions.d/common.d/n_network-functions.sh`

Function signature: cde74a72e94b4ca270a63fc91054620090ca6a1841b86382e554ab03431369eb

3.0.253 Function overview

The `n_network_get_interfaces` function is a Bash function that's used to discover the attributes of all network interfaces on a Linux system. It handles each network interface found in the `/sys/class/net` directory, skipping non-physical ones. For each valid interface, it gathers basic attributes like the operational state, MAC address, MTU, speed, driver name, IPs and IP assignment type (dhcp or static). The function then echoes this information in a string format.

3.0.254 Technical description

- **name** : `n_network_get_interfaces`
- **description** : Retrieves attributes of all network interfaces on a Linux system.
- **globals** : None.
- **arguments** : No arguments are passed to the function.
- **outputs** : For each network interface, outputs a string containing the interface's name, operational state, MAC address, MTU, speed, driver, IPs and IP assignment type.

- **returns** : Does not explicitly return a value.
- **example usage** : `n_network_get_interfaces`

3.0.255 Quality and security recommendations

1. Implement error-handling. Right now, the function does not handle potential errors beyond suppressing the output of commands that might fail. Improving error-handling could make the function more robust and easier to debug.
2. Validate inputs. While the function currently doesn't accept any arguments, if this changes in the future, all user-supplied inputs should be validated.
3. Keep up-to-date with system changes. The function relies on several system files and commands to retrieve network interface information. Ensure these are consistent with the current system configuration.
4. Avoid exposing sensitive information. The `echo` command used in the function could potentially reveal sensitive information (such as MAC addresses), depending on where the function's output is directed.
5. Simplify the function for better maintainability. The function is quite long and complex, try to simplify and refactor the function to make it easier to read and maintain.

3.0.256 `n_network_select_storage_interface`

Contained in `node-manager/base/n_network-storage-functions.sh`

Function signature: 31afa0803d6c2379fb602dea003ab030c79840eb8d4986055f54e13b7edef237

3.0.257 Function Overview

The Bash function `n_network_select_storage_interface()` is a networking tool that attempts to identify the most efficient network interface available and returns this to the user. It first looks for interfaces with a speed rating of 10Gb/s or higher, then defaults to any available speed should a faster connection not be present. The function also clears any extraneous whitespace from the returned network interface information.

3.0.258 Technical Description

- **Name:** `n_network_select_storage_interface()`
- **Description:** This function seeks to identify the best network interface available. It first tries to find an interface with a speed of 10G+ and if that doesn't yield any results, it will fallback to any speed. It then cleans up any excess whitespace before echoing the chosen interface to standard output.
- **Globals:** None
- **Arguments:** None

- **Outputs:** Echoes the chosen interface to standard output.
- **Returns:** Returns “0” if it finds a valid network interface, otherwise returns “1”.
- **Example usage:** `myInterface=$(n_network_select_storage_interface)`

3.0.259 Quality and Security Recommendations

1. Introduce error checking to handle scenarios where function `n_network_find_best_interface` is not available or does not work as expected.
2. Include explicit checks for null or unexpected values of the variable `$iface`.
3. Incorporate logs or a verbose mode into the function to help any future debugging efforts, making it easier to find potential issues or inefficiencies within the function processing.
4. Establish usage of more secure shell commands or functions where applicable to mitigate the potential risk of shell command injection.
5. Implement unit testing with a specific set of expected inputs and outputs to ensure the function behavior is as expected at any given time and to promptly catch any behavioral changes.

3.0.260 `n_network_show_vlans`

Contained in `lib/node-functions.d/common.d/n_network-functions.sh`

Function signature: f26ff21c1e6be88c21597a853a7bba0ce26eb5cdde8d64ee5d8674a12a3b2bbc

3.0.261 Function Overview

This function, `n_network_show_vlans`, explores the `/sys/class/net` directory. It attempts to find network devices that have VLANs (Virtual Local Area Networks). For each valid VLAN device, it collects various statistics about the device such as its current operational state, MTU (Maximum Transmission Unit), VLAN ID, and any IPv4 addresses associated with it.

3.0.262 Technical Description

This function has the following technical details:

- **Name:** n_network_show_vlans
- **Description:** Iterates over the `/sys/class/net` folder directory. It checks for network interfaces that carry VLANs. For each valid interface, it captures the interface’s current operational state, MTU, VLAN ID, and associated IPv4 addresses.
- **Globals:** None.
- **Arguments:** None.

- **Outputs:** For each valid network device, it echoes a colon-separated string containing the `interface name`, `operational state`, `IPv4 addresses`, `MTU`, and `VLAN ID`.
- **Returns:** It doesn't explicitly return a value. However, it echoes the required string for each valid network interface.
- **Example usage:**

  To show VLANs in a network, simply call the function without any argument as `n_network_show_vlans`.

3.0.263 Quality and Security Recommendations

1. Add more error handling: The script currently defaults to continue on errors such as invalid interface entries in the `/sys/class/net` directory. Consider adding more granular error checking and reporting.
2. Use functions for operations performed repeatedly: For instance, the code to read from sysfs files could be collected into a single, reusable function with good error handling.
3. Validate directory and file paths: Current implementation may break if, during runtime, the expected interface directories or their contents are not available. Consider protobuf compatibilities.
4. Implement function return codes: To ensure that the function can be safely used in other scripts, consider including return codes to indicate the success or failure of the function.
5. Secure Information: Ensure information such as IP addresses, VLAN ID are used or stored securely when using this function as bash scripts are plain text files.

3.0.264 `n_node_information`

Contained in `lib/node-functions.d/common.d/common.sh`

Function signature: 74929d3d0eddb047d5e64379e90dffc1956cc56b3f72b1d2b31d9b0358c73d56

3.0.265 Function overview

The `n_node_information()` function in Bash is designed to scan a server system's information and display the vital elements in a clean, formatted manner. This includes aspects such as host configuration, IP, Gateway, Domain, Mac address, Uptime, operational services, state of the console, virtualization status and if the system has been recently updated. The function performs error checking on loading the host configuration and if the function fails, it notifies the user with an error message.

3.0.266 Technical description

- **Name**: `n_node_information`
- **Description**: This function retrieves specific details of a Linux host, formats and visualizes them in a clear and concise manner for the user. It loads the host configuration, collects a series of system and network data, checks the console status, counts active services, clears the screen if running interactively and displays the collected information. The function monitors console status and displays appropriate footers.
- **Globals**: `HOSTNAME`, `TYPE`, `HOST_PROFILE`, `STATE`, `IP`, `NETMASK`, `provisioning_node`, `mac_address`, `dns_domain`, `uptime_display`, `active_count`, `virtualization_status`, `virtualization_type`, `console_status`, `UPDATED`.
- **Arguments**: There are no explicit arguments used in this function.
- **Outputs**: Displays a system's information in the console, formatted in an organized format.
- **Returns**: If it fails to load the host configuration, it returns 1. If all lines of codes are executed correctly, it returns 0.
- **Example usage**: To use this function, simply call it in your script as `n_node_information`

3.0.267 Quality and Security Recommendations

1. As the function involves displaying sensitive system information, it should adequately handle access permissions and restrict unauthorized access.
2. Check inputs for all externally supplied data to ensure data is as expected and guard against command injection attacks.
3. To enhance readability and maintainability, consider adding more comments to complex code blocks.
4. To ensure quality, consider writing unit tests to cover every possible case.
5. Consider having error checking for each local variable where a remote command is used to fetch system information.
6. Encapsulate echo statements formatting the output within a separate function for maintainability and reuse.

3.0.268 `node_lio_create`

Contained in `node-manager/rocky-10/iscsi-management.sh`

Function signature: 98eefecb076b12ce8574e70fbcdc3dbe7da279292f2b27fad96ad89b969a1f6e

3.0.269 Function overview

The `node_lio_create` function in Bash is designed to configure a Linux node as a target for the iSCSI protocol. It does this by creating a block-level backstore for a specific

device, establishing an iSCSI target using a given IQN, building a LUN for the target, and, if specified, setting an ACL for a providing initiator. If any step fails, it aborts the operation and reverts any successful changes made prior to the error. The function also toggles off authentication for testing and demo purposes if no ACL is provided.

3.0.270 Technical description

3.0.270.1 Name

`node_lio_create`

3.0.270.2 Description

This script is created to configure a Linux node as a target of the iSCSI protocol. It creates a backstore for a certain device, creates an iSCSI target using a specified iqn, creates a LUN for the target and sets an ACL for an initiator (if provided). If any step in the process fails, the function reverts all successful steps completed prior to the failure. The function also turns off authentication if no ACL is provided for testing and demo purposes.

3.0.270.3 Globals

None

3.0.270.4 Arguments

- `$1` (`--iqn`): The IQN to be used to create the iSCSI target
- `$2` (`--device`): The device for which the backstore is created
- `$3` (`--acl`): The ACL for the initiator (optional)

3.0.270.5 Outputs

Logs describing what the function is currently doing, whether it is successful, and any errors encountered.

3.0.270.6 Returns

- `0`: If the function runs successfully
- `1`: If any of the steps (parsing arguments, creating a backstore, creating a target, etc.) fail

3.0.270.7 Example usage

```
node_lio_create --iqn iqn.2003-01.org.linux-iscsi.localhost.x8664:sn.42b456cd3f
--device /dev/sdb --acl iqn.1994-05.com.redhat:rhel7
```

3.0.271 Quality and security recommendations

1. Use more secure methods when demonstration mode is not required. The current setup allows writing access for any initiator, which could lead to unauthorized data modification.
2. Improve error handling to give more specific error messages, especially when parsing arguments.
3. Ensure that essential input is validated for correct format to prevent any injection attacks or unexpected errors. For instance, validate the IQN structure and the physical existence of the device.
4. Separate the aspects of the function into smaller, specific functions. This not only makes the function easier to read and maintain, it also aids in debugging and testing.
5. Add input sanitation for all user-provided inputs. Bash functions can be prone to injection attacks, including command injection, if inputs are not properly sanitized.

3.0.272 `node_lio_delete`

Contained in `node-manager/rocky-10/iscsi-management.sh`

Function signature: 198653d7f8219c254d0f7d905c865d8994dd5dc9072af9cdbc979296660c9662

3.0.273 Function overview

The function `node_lio_delete` is used to delete an iSCSI target and its associated backstore. The function works by parsing arguments to find the “iqn”, validating if the iqn is provided or not, extracting the backstore name from said IQN, checking if the target and backstore exists, and then deleting the iSCSI target and its backstore on confirmation.

3.0.274 Technical description

- **Name**: `node_lio_delete`
- **Description**: This function deletes an iSCSI target and its associated backstore using the iSCSI Qualified Name (IQN).
- **Globals**: N/A
- **Arguments**:
  - `$1: --iqn`: the iSCSI Qualified Name (IQN)
- **Outputs**: Logs to the remote log file
- **Returns**:
  - `0` if the iSCSI target and backstore are successfully removed
  - `1` if required parameters are missing, or if the target and/or backstore do not exist or fail to be deleted.
- **Example usage**:
  `node_lio_delete --iqn iqn.2003-01.org.linux-iscsi.localhost.x8664:sn.4cd98b1141`

3.0.275 Quality and security recommendations

1. Validate the argument to ensure it is a well-formed IQN before attempting to process it.
2. Add error handling to address possible failure conditions when interacting with `targetcli`.
3. Use local variables wherever possible to avoid potential data leakage or accidental overwriting.
4. Use `local -r` to declare constants which prevents any subsequent code from accidentally modifying the variable.
5. Document each component of the function, and explain the purpose behind every major decision. This would be beneficial for maintainability and readability.

3.0.276 `node_lio_list`

Contained in `node-manager/rocky-10/iscsi-management.sh`

Function signature: f36324bce9bac497425dd3ef1f0cb414f4b7fe0e640904c41e749f78c8e29a70

3.0.277 Function overview

The `node_lio_list` is a Bash function designed to provide information about iSCSI targets and block backstores, which are fundamental components in storage network protocols using iSCSI.

3.0.278 Technical description

- **name:** `node_lio_list`
- **description:** This function lists and echoes the iSCSI targets and block backstores by utilizing the `targetcli` command, a management shell for Linux-IO (LIO) and target subsystems of the Linux kernel. It first echoes a header, runs the `targetcli /iscsi ls` command to list iSCSI targets, gives a line break, echoes another header, runs the `targetcli /backstores/block ls` to list block backstores, and then returns a 0 to indicate the function ran successfully.
- **globals:** None.
- **arguments:** None.
- **outputs:** The list of iSCSI targets and block backstores.
- **returns:** 0 (indicating the function has completed successfully).
- **example usage:**

```
To invoke the function, simply call it as follows:
node_lio_list
```

3.0.279 Quality and security recommendations

1. It is highly recommended to use local variables inside the function rather than global variables to avoid possible conflicts with other scripts or processes.
2. It would be best to handle the possible command errors, e.g., by capturing and checking the exit status of the `targetcli` commands to provide even higher reliability.
3. Adding a function comment at the start outlining what the function does is good practice for code clarity and maintainability.
4. It is recommended to verify if `targetcli` tool is installed before attempting to run the commands.
5. Consider adding user input validation if there were to be any arguments or parameters in the future.
6. To further enhance the security aspect, ensure that the necessary and appropriate permissions (like sudo permissions for the `targetcli` command) are in place before the script can be executed.

3.0.280 `node_lio_manage`

Contained in `node-manager/rocky-10/iscsi-management.sh`

Function signature: 86138e13e14957c8703d9fb6016f95dbd6eec12c63716fc419c9afcb1c48e6ca

3.0.281 Function overview

The `node_lio_manage` is a function in Bash that handles operations related to I/O nodes. It takes an action and an optional parameter as arguments. The action can be any of the following: create, delete, start, stop, status, or list. If the function encounters an unsupported action, it logs a warning message and returns 1.

3.0.282 Technical description

- **Name:** node_lio_manage
- **Description:** This function manages operations of I/O nodes. It can create, delete, start, stop, provide status, or list based on the action argument it receives.
- **Globals:** None
- **Arguments:**
  - $1: This is the action to perform. It could be 'create', 'delete', 'start', 'stop', 'status', or 'list'.
  - $2: This should contain extra options or arguments, if any, required for the action.
- **Outputs:** This function outputs different responses based on the action it is performing. For invalid or missing action, it logs a warning message.
- **Returns:** It returns 1 if the action is invalid or missing, and returns the status code of the executed action otherwise.

- **Example usage:** `node_lio_manage create [options]`

3.0.283 Quality and security recommendations

1. The input arguments, especially the 'action' argument, should be sanitised to prevent command injections.
2. The function does not have global dependencies, which is a good coding practice. It's always better to avoid global variables as much as possible.
3. Use more descriptive error messages that can help the user when invalid or missing arguments are provided.
4. When returning the status code of the executed action, ensure that the possible status codes and their meanings are well documented.
5. Always perform checks for any potential null or undefined values in the arguments to make the function robust and error-free.

3.0.284 `node_lio_start`

Contained in `node-manager/rocky-10/iscsi-management.sh`

Function signature: 618a6b47c51365d7704455f712c3beca1d289fd4105a628552ebf67c3fbc5573

3.0.285 Function Overview

`node_lio_start` is a bash function which is used to start a system service, specifically the `target` service. The function first logs the intent to start the service, thereafter using systemctl to initiate the start process. If the service starts successfully and is enabled, the function logs a success message and returns a 0. If unsuccessful, the service logs a failure message and returns a 1.

3.0.286 Technical Description

Here is a detailed description block for this function:

- **Name**: node_lio_start
- **Description**: Initiates the startup of the 'target' service using 'systemctl'. If the service starts and is enabled successfully, the function logs a success message and returns '0'. If the startup fails, the function logs a failure message and returns '1'.
- **Globals**: None
- **Arguments**: None
- **Outputs**:
  - Logs "Starting target service"
  - If successful, logs "Successfully started target service"
  - If unsuccessful, logs "Failed to start target service"
- **Returns**:
  - 0: If the startup of the service is successful

  - 1: If the startup of the service fails
- **Example usage**: `node_lio_start`

3.0.287 Quality and Security Recommendations

1. Include error handling mechanism for the invocation of 'systemctl'. Currently, the function assumes 'systemctl' will always execute without error. An improvement would be to handle any potential error from the 'systemctl' command.
2. Take the service name ('target') as argument instead of hardcoding it. This would make the function more reusable and generic.
3. Consider encapsulating logging functionality into a separate function to promote code reuse and separation of concerns.
4. For security reasons, consider checking permissions before trying to start the service as certain services may require administrator or other elevated permissions.

3.0.288 `node_lio_status`

Contained in `node-manager/rocky-10/iscsi-management.sh`

Function signature: 327a33e59e304acd75f9e12b0cd6fa4aca564708c04f3615da39246c5573ba39

3.0.289 Function overview

The function `node_lio_status` performs two primary tasks: it checks the status of a 'Target Service' and it lists the LIO (Linux-IO) configuration information. It does this by leveraging the `systemctl` and `targetcli` commands appropriately. The function is useful for quickly acquiring information relevant to the status and configuration of Linux input/output operations.

3.0.290 Technical description

```
- name: node_lio_status
- description: Node Linux-IO (LIO) Status displays the status of
 ↪  the "Target Service" and lists the current LIO configuration
- globals: None
- arguments: None
- outputs: Prints to standard output the status of the 'Target
 ↪  Service' and the current LIO configuration
- returns: 0 (Zero, indicating that the function has executed
 ↪  successfully)
- example usage: node_lio_status
```

3.0.291 Quality and security recommendations

1. Introduce error handling: Currently, the function does not consider cases where the `systemctl` command fails to execute or where the 'Target Service' does not

exist. Implementing error handling would increase the function's robustness.

2. Validate permissions: Validate that the user has appropriate permissions before utilizing `systemctl` or `targetcli`. This would help prevent potential security issues.
3. Improve function documentation: Include comments within the function to explain what individual commands are doing. This will make the function more understandable to others, improving maintainability.
4. Consider return values: While this function always returns 0 currently, it might be more informative if it could return different values based on the status of the 'Target Service' or the LIO configuration, which would provide more valuable information to the users or scripts using this function.

3.0.292 `node_lio_stop`

Contained in `node-manager/rocky-10/iscsi-management.sh`

Function signature: 971a38bb22f277cd0531cfa279368d8f7e4995e9a385bb6221485f8a90cc3bcd

3.0.293 Function overview

This `node_lio_stop` function is designed to stop a specific system service named `target`. It first logs to a remote system that it is attempting to stop the service. It then tries to stop the service using the `systemctl stop` command. If the service is stopped successfully, it logs a success message to the remote system and returns a `0`. If it fails to stop the service, it logs a failure message and returns a `1`.

3.0.294 Technical description

Definition block:

- **name**: `node_lio_stop`
- **description**: This function attempts to stop a specific system service named `target`. It logs to a remote system about the start and end of the operation, whether successful or not.
- **globals**: `remote_log`: A function that helps log messages to a remote host.
- **arguments**: None
- **outputs**: Logs to a remote system about the attempt to stop and the result of stopping the service, either successful or failed.
- **returns**: `0` on successful stoppage of the service, `1` on failure.
- **example usage**: `node_lio_stop`

3.0.295 Quality and security recommendations

1. Always make sure the `remote_log` function is secured and only authorized systems have access to push logs.
2. One should verify whether the `systemd` service named `target` is present on the machine before attempting to stop it.
3. Ensure proper error handling is in place, e.g., if the `systemctl` command fails to run due to insufficient privileges.
4. It would be more efficient to specify the service to stop as a parameter, rather than hard-coding `target` into the script.
5. It may be beneficial to implement some form of backup or verification before stopping services if the service is critical.
6. Ensure this script is only run by authorized users or systems with the necessary permissions to avoid misuse.

3.0.296 `node_storage_manager`

Contained in `node-manager/base/n_storage-functions.sh`

Function signature: 739f69b5a6036d980bc51405a34e0ec8a34bed5236ca2984db5c0d656a80c5c6

3.0.297 Function overview

The `node_storage_manager` function in Bash is a utility for managing storage components. It validates the input arguments and dispatches the appropriate management command to a specific storage component, i.e., either “lio” or “zvol”. Upon completion or failure, it logs an appropriate message to indicate either success or failure and provides the error code (if any).

3.0.298 Technical description

- **Name:** `node_storage_manager`
- **Description:** Executes a command for a given component and logs the resulting status.
- **Globals:** None
- **Arguments:**
  - `$1 (component)`: The component to be managed.
  - `$2 (action)`: The action to be executed on the component.
  - `[options]`: Additional options for the action.
- **Outputs:** Logs a message indicating either successful or failed execution.
- **Returns:** If the execution is successful, it returns `0`. In case of a failed execution, it returns the non-zero error code.

- **Example usage:**

```
node_storage_manager lio create
node_storage_manager zvol delete
```

3.0.299 Quality and security recommendations

1. Thoroughly validate all command inputs to avoid command injection or other types of malicious attacks.
2. Implement a stricter policy for deciding which users can execute these commands. Explicit access control helps in reducing unauthorized system changes.
3. Include more detailed logging. Gaining insights into the sequence of operations performed by the script is better for debugging and auditing.
4. Use encrypted channels for remote logging to protect sensitive information.
5. Always handle error cases and provide appropriate responses to calling functions. This prevents propagation of errors.

3.0.300 `node_zvol_check`

Contained in `node-manager/rocky-10/zvol-management.sh`

Function signature: 07447af71573f33e1e9896011244fa50252bca03c45d1685a0b8677a4720d46c

3.0.301 Function overview

`node_zvol_check` is a bash function that checks the existence of a given zvol (ZFS volume) inside a provided ZFS pool. The function takes two arguments: '–pool' followed by the name of the ZFS pool and '–name' followed by the name of the zvol. It relies on the `zfs list` command to determine the presence of the zvol and utilizes a `remote_log` function to record the actions taken and their result.

3.0.302 Technical description

- **Name:** `node_zvol_check`
- **Description:** This function confirms the existence of a provided zvol in a specified ZFS pool. Returns 1 if the zvol does not exist or if required parameters are missing and 0 if the zvol is present.
- **Globals:** None.
- **Arguments:**
  - `$1`: Should be `--pool`. The name of the ZFS pool is provided as the next argument.
  - `$2`: Shoud be `--name`. The name of the ZFS volume is provided as the next argument.

- **Outputs:** This function logs messages to the `remote_log` function, which could print to stdout, stderr, or another location depending on the implementation of that function.
- **Returns:** 0 if the zvol is present. 1 if the zvol does not exist or if required parameters are missing.
- **Example Usage:**

```
node_zvol_check --pool zroot --name tank
```

3.0.303 Quality and security recommendations

1. Consider implementing error checking for the `zfs` command to handle potential failures (eg. absent `zfs` command or insufficient user permissions).
2. Secure the `zfs` command execution by specifying explicit paths, ensuring that there are no malicious alternatives in the PATH.
3. Properly sanitize and check input to prevent potential command injection vulnerabilities.
4. Since the function depends on the `remote_log` function, ensure that it doesn't leak sensitive information and is adequately secured against potential threats.
5. Utilize secure coding practices and regularly review the code to mitigate other potential security issues.

3.0.304 `node_zvol_create`

Contained in `node-manager/rocky-10/zvol-management.sh`

Function signature: ec6e430fb8789091cfe8fdd4b1e197a7a20d2619e9cb9986827d740d74268de1

3.0.305 Function Overview

This function, `node_zvol_create`, creates a ZFS volume (zvol) on a ZFS storage pool. It parses the incoming arguments for the name of the pool, name of the volume, and desired size of the volume. Then it validates these parameters and creates the zvol if it doesn't already exist on the specified pool.

3.0.306 Technical Description

- **Name**: `node_zvol_create`
- **Description**: The function creates a zvol on a ZFS storage pool, based on the pool, name, and size arguments.
- **Globals**: None.
- **Arguments**:
  - `$1` (`--pool`): Name of the ZFS storage pool.
  - `$2` (`--name`): Name of the zvol to be created.
  - `$3` (`--size`): Size of the zvol to be created.

- **Outputs**: Logs information about the operation status, like parameter parsing, result of zvol creation, etc.
- **Returns**:
  - 0 if the zvol is created successfully.
  - 1 if any error occurred: invalid parameter, missing required parameters, or failed to create the zvol.
- **Example usage**:

```
node_zvol_create --pool tank --name vol01 --size 10G
```

3.0.307 Quality and Security Recommendations

1. Parametrize the error messages to avoid duplications and improve maintainability.
2. Enhance parameter validation to include checks for the validity of the pool, the validity of the size, etc.
3. Use more specific return codes for various error conditions. This can help the calling code in identifying the specific error.
4. Log error messages to stderr to make it easier to distinguish between normal operation logs and error messages.
5. To prevent potential command failure, ensure that the ZFS utilities are installed and the user has sufficient privileges to create volumes.
6. Document the function, its usage, and its return and error codes in a developer-facing document to aid in its correct usage.

3.0.308 `node_zvol_delete`

Contained in `node-manager/rocky-10/zvol-management.sh`

Function signature: 21fe8b803dc5e01613ebe91d459c8ff30011593e6a87412bc6122f867aeec227

3.0.309 Function overview

The `node_zvol_delete()` function in Bash is designed to delete a specified ZFS volume (zvol) on a given pool. It parses and validates the necessary arguments (pool and name) and sends error messages through `remote_log` function in the event of missing or unknown parameters. It then checks if the specified zvol exists in the given pool before attempting to delete it, returning a success message through `remote_log` function if the deletion is successful or an error message if it fails.

3.0.310 Technical description

Name: `node_zvol_delete()`

Description: Deletes a specified ZFS volume from a specified pool in Bash.

Globals: None

Arguments: - `$1` (`--pool`): name of the pool where the ZFS volume exists - `$2` (`--name`): name of the ZFS volume to be deleted

Outputs: Error or success messages are outputted through `remote_log` function.

Returns: - `0` if the zvol is successfully deleted. - `1` if there's an error in arguments or the zvol cannot be deleted.

Example usage: `node_zvol_delete --pool POOL_NAME --name ZVOL_NAME`

3.0.311 Quality and Security Recommendations

1. Add more detailed error descriptions to specify whether the pool or the zvol name was not provided.
2. Add input data verification to validate the pool or zvol does indeed exist before any operations.
3. Ensure that the use `remote_log` function for logging does not expose any sensitive info.
4. Add error handling or logging for the potential possibility of `zfs destroy` operation failure.
5. Include a function help or usage manual that can be invoked when needed.

3.0.312 `node_zvol_info`

Contained in `node-manager/rocky-10/zvol-management.sh`

Function signature: 80d5ff3413c64a25ffdb5394756a3fcb79290986c8d4a3bfd18cdf837a95a543

3.0.313 Function overview

The function `node_zvol_info` gathers information about a specific ZFS volume within a pool on a system. The function accepts two arguments, `--pool` and `--name`, specifying the pool and the volume name, respectively. After validating these arguments, the function checks if the specified volume exists. If the volume exists, the function displays information such as the name, volume size, used and available space, and the amount of disk space referenced. The function also prints the device path of the specified volume. The function uses the `zfs list` command to gather information about the volume.

3.0.314 Technical description

- **Name**: `node_zvol_info`
- **Description**: This function gathers and prints information about a specified ZFS volume.
- **Globals**: None
- **Arguments**:

  - `--pool`: The pool that the ZFS volume is in.
  - `--name`: The name of the ZFS volume.
- **Outputs**: If successful, prints information about the ZFS volume, such as its name, size, used and available space, the amount of referenced space, and the device path.
- **Returns**: The function returns 1 if it encounters an unknown parameter, if the `--pool` or `--name` parameters are not specified, or if the specified ZFS volume does not exist. If successful, the function returns 0.
- **Example Usage**: `node_zvol_info --pool tank --name comments`

3.0.315 Quality and security recommendations

1. Implement stricter validation for arguments to prevent possible misuse or erroneous calls to the `zfs list` command.
2. Provide error handling that not only logs the error but also informs the caller about the issue in a structured manner.
3. The printed output could potentially contain sensitive information, such as disk usage. Consider an option to obfuscate or exclude certain pieces of information.
4. Add functionality to handle permissions and to check if the current user has the necessary permissions to execute the `zfs list` command.
5. Make sure the function handles case-sensitive input correctly for the `--pool` and `--name` arguments.

3.0.316 `node_zvol_list`

Contained in `node-manager/rocky-10/zvol-management.sh`

Function signature: 95f86f7af006e06fa8df84870e8f7eff728b79dd9bcb448fd864b4d7b350276b

3.0.317 Function overview

The `node_zvol_list()` function is a bash shell function designed primarily to manage and list ZFS volumes or zvols in a specified or default pool stored in the `pool` variable. The function parses the arguments passed to it using a loop and case statement and lists the volumes in the specified pool or all volumes if no pool is specified. If an error occurs while listing zvols, the function logs the error with the `remote_log` function and returns an error code.

3.0.318 Technical description

Function definition:

- **Name:** node_zvol_list

- **Description:** Parses input arguments to identify the ZFS pool to list volumes from, and lists volumes of the specified ZFS pool. Logs errors and returns error codes accordingly.
- **Globals:** None
- **Arguments:**
  - $1: command line arguments, unknown or –pool. Unknown argument will trigger error reporting and halt execution.
  - $2: name of the ZFS pool to list volumes from, required if $1 is –pool.
- **Outputs:** lists the name and volume size of each volume in the specified ZFS pool.
- **Returns:**
  - 1 - If the function encounters an unknown argument or fails to list zvols.
  - 0 - If the function successfully lists zvols.
- **Example usage:** `node_zvol_list --pool poolname`

3.0.319 Quality and security recommendations

1. Implement input validation for the pool name to ensure it adheres to expected formatting and naming conventions of ZFS pools to avoid exploit attempts or errors.
2. Instead of suppressing errors (`2>/dev/null`), consider handling and logging them appropriately for better troubleshooting capabilities.
3. Consider limiting the list output to essential information for better clarity.
4. Use detailed, descriptive log messages that provide more context when errors occur.
5. Use consistent naming conventions and comments to improve maintainability and readability of the code.

3.0.320 `node_zvol_manage`

Contained in `node-manager/rocky-10/zvol-management.sh`

Function signature: fbc75475f9ed8b4e90c9fe423b227bbd08f041872fcaf46c3bee9ac215f09937

3.0.321 Function Overview

The `node_zvol_manage` is a Bash function designed to manage the zvol (ZFS volume) of a node in a system. This function receives an action as argument which it expects to be one of the following: create, delete, list, check, or info. When the function is invoked with a valid action, it associates the action with the relevant controller function such as `node_zvol_create`, `node_zvol_delete`, and so on. If an unrecognized action is passed, an error message will be logged and the function will exit with a return status of 1.Also, if the action argument is not provided, an error message will be logged specifying the function usage.

3.0.322 Technical Description

- **Name:** `node_zvol_manage`
- **Description:** This function manages the ZFS volumes (zvol) in a node. It validates the received action argument and dispatch it to the appropriate controller function.
- **Globals:** None
- **Arguments:**
  - `$1: action` - The action to be performed by the function. Valid actions are: create, delete, list, check, info.
- **Outputs:** Logs information / errors related to the processing of the function.
- **Returns:** 1 - if the action argument is not provided or if an unrecognized action is passed, 0 - otherwise.
- **Example Usage:** `node_zvol_manage create` - This will create a new zvol.

3.0.323 Quality and Security Recommendations

1. Validate other arguments along with the action argument. This may include checking if the values passed are in expected formats or within an appropriate range of values.
2. Implement error handling for each case action. Currently, if any of the `node_zvol_` functions fail, `node_zvol_manage` will not capture the error.
3. Log more detailed information at each step of the function for enhanced troubleshooting and robustness.
4. Consider adding a "help" action that can display more detailed information about how to use the function and what each action does.
5. Avoid showing full usage details in error messages as it might help an attacker identify ways to exploit the system.

3.0.324 `n_opensvc_join`

Contained in `lib/node-functions.d/common.d/n_opensvc-management.sh`

Function signature: 41b6e692c5ecee93e0e786fa6c67df0b0729b5352917c53d8e4d9f841a3f3765

3.0.325 Function overview

The `n_opensvc_join()` function in a bash script is designed to automate the process of joining an OpenSVC cluster. This function first sets a node and obtains an authentication token. Then, it attempts to join the cluster and logs the result. If the cluster joining fails, it logs an error message and returns 1. If the joining process is successful, it logs a success message, updates a remote host variable to record the current time (in seconds) indicating the completion of the join, and returns 0.

3.0.326 Technical description

Function Details:

- **name:** n_opensvc_join
- **description:** This function aims to facilitate the process of joining an OpenSVC cluster by performing an automated join attempt.
- **globals:**
  - `osvc_node: This variable holds the node name.`
  - `osvc_token: This value stores the authentication token required in the OpenSVC cluster join command.`
  - `VAR: This variable temporarily holds command output.`
- **arguments:**
  - `There are no arguments passed directly to this function.`
- **outputs:**
  - `Joining cluster node: This log message displays the node that's being joined.`
  - `Failed to join cluster: Logs this error message if the join fails.`
  - `Join command completed: Logs this success message if the join is successful.`
  - `Updating host variable cluster_joined with timestamp (in seconds): This action occurs after the successful joining of a cluster.`
- **returns:**
  - `Returns 1: If it fails to join the cluster.`
  - `Returns 0: If the joining process is successful.`
- **example usage:** `n_opensvc_join`

3.0.327 Quality and security recommendations

1. Ensure variable values are sanitized to prevent command injection attacks.
2. Handle all potential error cases and ensure they return appropriate error codes.
3. Store the authentication token securely to prevent credential exposure.
4. Validate a successful join with more than just a command's exit status for increased reliability.
5. Implement logging for all important steps to facilitate debugging.
6. Include detailed comments to increase code readability and maintainability.

3.0.328 `n_osvc_start`

Contained in `lib/node-functions.d/common.d/n_opensvc-management.sh`

Function signature: c7c58fcf02e876ead3044a939ba5028773fba0e5e81971751e0c72ce1c1f4b0b

3.0.329 Function Overview

This function, `n_osvc_start()`, is designed to start the OpenSVC server. Initially, it prevents the default rc (run commands) from running and notifies that it is starting the OpenSVC through a remote log. Thereafter, it initiates a daemon processed in the background, which is redirected to the system log with a debug priority. It then waits for the socket to establish a connection and once done, it logs that the OpenSVC daemon has successfully started.

3.0.330 Technical Description

- **Name**: `n_osvc_start`
- **Description**: This function is meant to initialize the OpenSVC server. It first prevents the default rc from operating, then logs the initiation process, runs the daemon in the background to be logged with a debug priority, waits for the socket, and finally logs a successful start.
- **Globals**: None.
- **Arguments**: None.
- **Outputs**: Logs from the function such as “Starting OpenSVC” and “OpenSVC daemon started (logging to syslog)” are generated and sent to `n_remote_log` function. Process output from the daemon is sent to system logger.
- **Returns**: Does not return any value.
- **Example usage**: `n_osvc_start`

3.0.331 Quality and Security Recommendations

1. Use encryption when communicating to the log server to ensure log data is securely transmitted.
2. Implement error handling to catch any failures during the process, and also log the error details for debugging purposes.
3. The function does not have any input validation. Please consider adding necessary validation to ensure function integrity.
4. This function relies heavily on system-level commands, which have the potential for command injection if not properly handled. As this function does not accept any arguments or parameterized inputs, the risk of injection is relatively low, but it’s something to consider when modifying or expanding this function.

3.0.332 `n_osvc_wait_for_socket`

Contained in `lib/node-functions.d/common.d/n_opensvc-management.sh`

Function signature: d9773f250590ec0655b45d26f074b1a33def4ce46a154c200cc7a154def06d0d

3.0.333 Function Overview

The function `n_osvc_wait_for_socket()` is designed to check and wait for an OpenSVC daemon socket to be prepared and ready. Initially, it applies a log entry stating that it is waiting for the daemon socket. Additionally, it sets up a local variable and implements a loop that checks if the socket is ready for 10 iterations. If the socket is detected as ready, the function puts out another log entry, then halts with a successful execution, in case the socket is not ready by the end of the iterations, the function signs off with a log entry displaying the function's failure to identify the ready socket, then exits with a failed execution.

3.0.334 Technical Description

```
Function: n_osvc_wait_for_socket
Description: The function checks if the OpenSVC daemon socket is
 ↪  ready for 10 seconds and logs messages based on the result. It
 ↪  waits for 1 second in each iteration and ends the execution
 ↪  with a success if it finds the socket to be ready, or with a
 ↪  failure if not.
Globals: No global variables are modified in this function.
Arguments: The function does not use any arguments.
Outputs: Standard Output and Standard Error are not used. All
 ↪  output messages are getting written to the log.
Returns: Returns 0 if the OpenSVC socket is ready, and 1 if the
 ↪  socket is not ready after 10 seconds.
Example Usage: n_osvc_wait_for_socket
```

3.0.335 Quality and Security Recommendations

1. Implement output validation to guarantee the accuracy of log entry outputs.
2. Increase the waiting time before each cyclic re-check, based on server performance, to optimize function efficiency.
3. Include error handling to handle potential failure points and unexpected behavior.
4. Implement additional logging measures to ensure detailed reports on potential errors, their locations, and causes.
5. Compose and execute scripts as the least privileged user to reduce potential harm from security breaches.
6. Keep the script up to date with the current Bash best practices for more stringent security measures.

3.0.336 `n_prepare_build_directory`

Contained in `lib/node-functions.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: 078b9d8f7b0fe1e5882492f4afbd27578cba41f46c064e5c5366cb46685a3cf4

3.0.337 Function overview

The function `n_prepare_build_directory()` is designed to prepare a build directory for a specific version of OpenSVC - an open-source IT service and resource manager. The function takes no arguments, and mainly works by setting up several global variables and checking certain conditions. The function returns `1` in case of error, and `0` upon successful completion.

3.0.338 Technical description

- Name: `n_prepare_build_directory` **- Description:** Prepares a build directory to install or update a specific version of OpenSVC. Checks for required global variables and previous actions, creates a temporary build directory, copies the source code into it, and sets up the build environment. **- Globals:** - `OPENSVC_VERSION`: The version of OpenSVC to be installed. - `OPENSVC_BUILD_DIR`: The directory where the build process will take place. - `CGO_ENABLED`: A flag indicating if the Go build environment is enabled for static compilation. **- Arguments:** None. **- Outputs:** Echoes messages about operation progress and errors. **- Returns:** - `1` if any error occurs. - `0` on successful completion. **- Example usage:**

```
source n_prepare_build_directory.sh
```

3.0.339 Quality and security recommendations

1. For security purposes, particularly since the application is creating a temporary directory, take all precautions to prevent symbolical linking attacks (symlink race conditions).
2. Increase the level of logging, such as indicating which action the script is currently performing, especially in long running scripts.
3. Utilize a centralized error handling mechanism to ensure that returned errors are correctly interpreted and communicated.
4. Use unambiguous variable names to simplify code readability and maintainability.
5. Practice defensive programming by adding checks to ensure files and directories exist before using them.
6. Make sure that all temporary files and directories are deleted after use to prevent unnecessary resource consumption.

3.0.340 `n_remote_cluster_variable`

Contained in `lib/node-functions.d/pre-load.sh`

Function signature: 5854f511031b80482db5a46d1b8c555332355ba0732b028b2262070ca9ee2f70

3.0.341 Function overview

The bash function `n_remote_cluster_variable` is used to set or get a variable on a remote cluster. This function uses `n_ips_command` to perform its operations. This function can set a value for a given name or it can retrieve the value for a given name. If no value is found or if `n_ips_command` fails, it logs the error using `n_remote_log`.

3.0.342 Technical description

Name: `n_remote_cluster_variable`

Description: This function is designed to set or get a variable value in a remote cluster through a command.

Globals: None used.

Arguments: - `$1: name`: Name of the variable on the remote cluster. - `$2: value`: Value to set for the variable on the remote cluster. This argument is optional.

Outputs: The function writes to stdout the value of the variable on the remote cluster or logs any error that has occurred during its operations.

Returns: The function returns `0` if the operations are successful, or exits with `exit_code` if the operations fail. If the attempted operation is a GET but the variable does not exist, it returns a custom error code: 4.

Example usage: `n_remote_cluster_variable "variable_name" "variable_value"` - Will set the value for the specified variable. `n_remote_cluster_variable "variable_name"` - Will get the value for the specified variable.

3.0.343 Quality and security recommendations

1. Use a secure way to transfer the variables to the remote cluster in the `n_ips_command`.
2. Include better error handling in the case `n_ips_command` is unable to create the necessary http codes.
3. To prevent injection, ensure the name and value of the variable is correctly sanitized before being passed to the `n_ips_command`.
4. It would be advisable to set more descriptive custom error codes to differentiate between different failure modes.
5. To make the code more maintainable, consider breaking this function into two separate ones - one for setting and one for getting the values.

3.0.344 `n_remote_host_variable`

Contained in `node-manager/base/pre-load.sh`

Function signature: a6de9d196034590447c28a847e7c53ad418f07edd5fc8991351efef77e5c985a

3.0.345 1. Function overview

The function `n_remote_host_variable` is designed to facilitate the operation of various remote host variables. It allows users to perform several operations such as Get, Set, and Unset the remote host variables according to the parameter passed. It also features error handling mechanism and custom error codes.

3.0.346 2. Technical description

- **Function Name**: `n_remote_host_variable`
- **Description**: This function is responsible for managing different operations on remote host variables. These operations can be to retrieve(Get), set, or unset a remote host variable.
- **Globals**: No global variables are directly interacted with in this function.
- **Arguments**: `$1: name of the remote host variable to be operated upon, $2: holds the value to be set for the variable or the '--unset' flag indicating that the variable should be unset.`
- **Outputs**: The function outputs the result of the operation on the console.
- **Returns**: The function returns 0 for successful operations, while it returns custom error codes in case of failures.
- **Example usage**:
  1. To set a value: `n_remote_host_variable variableName variableValue`
  2. To get a value: `n_remote_host_variable variableName`
  3. To unset a value: `n_remote_host_variable variableName --unset`

3.0.347 3. Quality and security recommendations

1. Use more meaningful variable names to increase code readability.
2. Incorporate additional error handling measures to gracefully handle any unforeseen errors that may not currently be accounted for.
3. Throughout the code, don't trust any user input without sanitization to avoid shell command injection.
4. Perform input validation on the name and value parameters.
5. Replace hardcoded error codes with named constants, for better maintainability and readability.
6. Keep adding and updating the inline comments to improve maintainability of the code.
7. On security standpoint, always check the permissions given to function on bash script, and use the principle of least privilege.

3.0.348 n_remote_log

Contained in node-manager/alpine-3/TCH/BUILD/run_osvc_build.sh

Function signature: 317073fa4f65bfad64e4bff81f2b2ac946e498b574a3003339e2749585412787

3.0.349 Function Overview

The function `n_remote_log()` is designed to assist with remote logging from a bash shell. It uses the `logger` command to process a log message, then echoes that message prefixed with "osvc build:". This is ideal for logging build messages in a remote build system.

3.0.350 Technical Description

3.0.350.1 Name

n_remote_log

3.0.350.2 Description

A bash shell function that uses logging to help manage build processes in remote systems.

3.0.350.3 Globals

None

3.0.350.4 Arguments

- $1: The logging statement or event description to be logged.

3.0.350.5 Outputs

The function outputs the provided message to the standard log and echoes the message on the console.

3.0.350.6 Returns

The function does not return any particular value.

3.0.350.7 Example Usage

```
n_remote_log "This is a sample log message."
```

This will log and print the message "osvc build: This is a sample log message."

3.0.351 Quality and Security Recommendations

1. Consider validating the argument. Implement a check for `$1` to ensure it is not empty or contains only acceptable characters
2. To improve the quality of logs, implement a log level argument, so that details can be logged at different levels like info, warning, error.
3. For security, use secure string handling methods to filter or escape strings prior to sending them to logger.
4. Add error handling for the logger command to make this function more reliable and easier to debug.
5. Ensure that the device where logs are sent to is secured and logs are only accessible to authorized personnel.

3.0.352 `n_rescue_chroot`

Contained in `node-manager/alpine-3/RESCUE/rescue-functions.sh`

Function signature: 36ab383ed0615efcf4262c808ab544c9907213dd972b4d37a4cb96622a6d792e

3.0.353 Function Overview

This function `n_rescue_chroot()` facilitates chroot into a system which is already installed. It primarily performs the following operations:

1. Logs the start of chroot process.
2. Verifies if '/mnt' is mounted, and if not, logs the error message.
3. Prepares the chroot environment by ensuring essential filesystems are mounted on '/mnt'.
4. Determines the shell to use based on existence and executability.
5. Prints out a message representing entry into the chroot environment.
6. Executes the chroot and logs the status.
7. On exit, cleans up bind mounts and logs completion of the process.

3.0.354 Technical Description

- **Name:** n_rescue_chroot
- **Description:** This function is used to safely chroot into a system that is already installed.
- **Globals:** None
- **Arguments:** None
- **Outputs:** Logs certain activities, provides error messages when required, cleans up bind mounts.
- **Return:** Returns 1 in case '/mnt' is not mounted, else returns 0.
- **Example usage:** Simply calling the function without any arguments as `n_rescue_chroot`.

3.0.355 Quality and Security Recommendations

1. Input validation: Although this function receives no arguments, it's reliant on the state of the environment. Validate the preconditions to ensure it doesn't work in a potentially harmful state.
2. Error handling: Implement more comprehensive error handling, e.g., when the mounting of essential filesystems fails.
3. File system safety: Consider double-checking that the filesystems to be mounted are as expected.
4. Use of sudo: Any inappropriate use of this function could potentially harm the system as it is used for modifying mounted filesystems. It is recommended to use this command with appropriate rights only.
5. It's recommended to remove all echo and substitute it with logging functionality of script for consistency.

3.0.356 `n_rescue_cleanup`

Contained in `node-manager/alpine-3/RESCUE/rescue-functions.sh`

Function signature: 1b08a28bfc2d05fe8dc5a06588d0b41d3ac5914b62017d8da2f93b2af383c8f3

3.0.357 Function overview

The `n_rescue_cleanup` function is an integral part of a Bash script that helps perform various clean up operations on disk partitions. It does so by parsing arguments, logging remote information, reading and validating disk configurations, performing safety checks on the disk, unmounting filesystems, stopping MD arrays, zeroing superblocks, wiping filesystem signatures, clearing host configurations, and sets the final state of the node based on user input.

3.0.358 Technical Description

- **Name**: `n_rescue_cleanup`
- **Description**: This function is designed to clean up disk partitions and associated configurations. This is typically used in system setup and recovery scenarios.
- **Globals**: `n_rescue_cleanup` does not appear to rely on any global variables.
- **Arguments**: `n_rescue_cleanup` takes the following optional arguments:
  1. `-f|--force`: Bypasses user interaction and confirmation prompts.
  2. `--wipe-table`: Wipes the partition table of the given disk.
  3. `[disk]`: The disk to cleanup
- **Outputs**: The function outputs logs and information to stderr throughout its execution. It will display information on operations such as unmounting filesystems, stopping MD arrays, wiping filesystem signatures, clearing host configurations, etc.
- **Returns**: The function will return numeric status codes corresponding to various exit points in the function, including `0` if everything completes successfully.

- **Example usage**: `n_rescue_cleanup /dev/sda` would be used to cleanup the disk at `/dev/sda`.

3.0.359 Quality and Security Recommendations

1. Handle sensitive data: Ensure usage of this function in a script does not risk exposure of sensitive data (e.g. machine names, IP addresses) in logs or error messages.
2. Error handling: Improve on error handling, ensuring all possible error cases are covered and the output is logged.
3. Input validation: Be cautious of data that is passed into the function. Verify the disk input argument before proceeding with the function execution.
4. Use `-e` bash flag: This function might benefit from the `-e` bash flag, which causes the shell to exit if any invoked command exits with a non-zero status.
5. User prompts: In scripts meant to run without human interaction, it would be beneficial to remove or bypass user confirmation prompts.
6. Clearer documentation: While the function is reasonably well-commented, it could be beneficial to provide an upfront comment block detailing the function, its arguments, return values, and side effects.
7. Environment independence: Rely on environment variables and configuration files instead of hardcoded values to make the script adaptable to different environments.

3.0.360 `n_rescue_configure_profile`

Contained in `node-manager/alpine-3/RESCUE/rescue-functions.sh`

Function signature: 26d18ec3799d9cb52b07a3b10523a89b31f2aacd2c9e43cb91389cc6d0846dde

3.0.361 Function overview

The function `n_rescue_configure_profile` is used to configure a rescue mode profile. Firstly, this function implements a log with the tag "[INFO] Configuring rescue mode profile". It then displays a configuration message and creates a directory `profile.d` under the path `/etc/`.

A new file `rescue.sh` is created under the `profile.d` directory and it's expected to run in interactive shells utilizing a safe runner to execute rescue functions. If the `rescue.sh` is not created successfully or cannot set permissions, it will log errors and return 1. Finally, the function verifies whether the created file is readable and logs its state.

3.0.362 Technical description

- **name:** `n_rescue_configure_profile`

- **description:** Configures a rescue mode profile. The profile is executed in interactive shells, uses safe runner to run rescue functions, and ensures the file 'rescue.sh' is created, permissions set, and verified for readability.
- **globals:** N/A
- **arguments:** N/A
- **outputs:** Logs the beginning of profile configuration, failure or success of file creation, setting permissions, and file readability. Also provides user readable console logs.
- **returns:** `1` if creation or permissions of file fails, otherwise `0`.
- **example usage:** `n_rescue_configure_profile`

3.0.363 Quality and security recommendations

1. Add more error handling and checks during the execution of the function to confirm if each stage is completed successfully.
2. Use comments to explain complex or important parts of the code to make it easily understandable for other developers.
3. Evaluate the necessity for command output redirection throughout the function and consider the circumstances where it is used. Ensure that no sensitive details are leaked and essential information is not discarded.
4. Always use absolute paths for directories or files to avoid ambiguity which can be a security risk.
5. Consider the correct and secure permissions for files and directories created to decrease potential attack vectors.

3.0.364 `n_rescue_display_config`

Contained in `node-manager/alpine-3/RESCUE/rescue-functions.sh`

Function signature: 8ca19767c5ae352ec70f1996f4bb563272e1c075093d8e519eafe7f6cf85a0d0

3.0.365 Function Overview

The `n_rescue_display_config` bash function is used to receive and display the disk configuration from the IPS (Intelligent Power Software). It reads the configuration from a `host_config` file, lists the configurations, reports if no disk configuration is found, advises on running `lsblk` or `fdisk -l` to explore available disks, checks if the configured devices actually exists and provides suggested mount commands if the devices exist.

3.0.366 Technical Description

- **name**: `n_rescue_display_config`

- **description**: This bash function reads and displays the disk configuration from IPS which is stored in a `host_config`. If no disk configuration is found, it issues a warning. It also verifies the existence of the configured root and boot devices and gives suggested mount commands if these devices exist.
- **globals**: [n_remote_log: This function is used to log messages remotely]
- **arguments**:
  - No arguments are being passed to this function.
- **outputs**: The function prints various messages to the standard error including the current disk configuration obtained from the IPS or warning in case of no configuration found or if configured devices do not exist.
- **returns**: Returns 1 if no disk configuration is found and 0 after successfully displaying the disk configuration.
- **example usage**: `n_rescue_display_config`

3.0.367 Quality and Security Recommendations

1. Consider using more descriptive variable names, which might make the script easier to read and maintain.
2. Ensure Security by restricting the file permissions of the `host_config` file, limit access to trusted users.
3. Integrity Checks should be added before reading configuration from the `host_config` file.
4. Adding more detailed logging might be useful for debugging. Ensure log files have the correct permissions and are stored safely.
5. Ensure the bash script is free from global variables to prevent accidental override.
6. A more robust error-handling mechanism may be considered.
7. Avoid using `eval` as it may open up the code for command injection attacks.

3.0.368 `n_rescue_exit`

Contained in `node-manager/alpine-3/RESCUE/rescue-functions.sh`

Function signature: 377eab0ae64d3cb0c0f6903f6c19d5e7f928eb0aef72bcf3f0d778d5b6acc408

3.0.369 Function Overview

The function `n_rescue_exit()` is used to exit from rescue mode in a system. The function performs the following operations: logs the exit operation, unmounts any file systems under /mnt, clears the RESCUE flag, and provides instructions for rebooting in various states depending on the success of the system fix.

3.0.370 Technical Description

- **Name**: `n_rescue_exit()`

- **Description**: This function is designed to exit a system from the rescue mode. It undertakes several operations, including logging the exit, unmounting mounted filesystems under /mnt, clearing the rescue flag and provides instructions for next steps, depending on whether a system fix was successful or not.
- **Globals**: No global variables.
- **Arguments**: The function does not take any arguments.
- **Outputs**: The function provides console outputs describing the steps being undertaken. These include messages for exiting the rescue mode, unmounting the filesystems, clearing the RESCUE flag and a set of instructions for rebooting in different states.
- **Returns**:
  - The function returns `0` if successful.
  - If it fails to clear the RESCUE flag, it returns `1`.
- **Example Usage**:

```
n_rescue_exit
```

3.0.371 Quality and Security Recommendations

1. Validate all inputs and include checks for different states and edge cases.
2. Always log events with as much detail as necessary, but without including sensitive information.
3. Use modern, secure methods for necessary tasks such as unmounting the filesystem.
4. Ensure error handling and logging are in place for fault detection.
5. Propagate error codes and handle them at the call site to prevent propagation of failure and easier debugging.

3.0.372 `n_rescue_fsck`

Contained in `node-manager/alpine-3/RESCUE/rescue-functions.sh`

Function signature: 8706a72f7f8678fb91586b882b1d36312e88abb735d9398c82f6e30c0b5ac178

3.0.373 1. Function Overview

The `n_rescue_fsck` function is designed to check and repair file systems on specific devices. It takes one argument, the target device, and use `e2fsck` utility to check the provided device. If no device is specified, it will ask for a device and list available devices. If the provided device is mounted, it will unmount it before the check. It also ensures the `e2fsck` utility is installed. The function interprets and logs the return code of `e2fsck`, providing meaningful feedback about the result.

3.0.374 2. Technical Description

- **Name:** n_rescue_fsck
- **Description:** This function is used for checking and repairing file systems on given devices.
- **Globals:**
  - **VAR:** n/a
- **Arguments:**
  - **$1:** The target device to check.
- **Outputs:** Logs containing check results.
- **Returns:**
  - **0:** No errors or errors corrected successfully.
  - **1:** If the given device does not exist or is not a block device.
  - **2:** If there were problems installing the `e2fsck` utility, or if the check process faced uncorrectable errors or operational errors.
- **Example usage:** `n_rescue_fsck /dev/sda1`

3.0.375 3. Quality and Security Recommendations

1. Make sure the function is only run with the necessary privileges to avoid unnecessary security risks.
2. Consider handling other types of file systems, not just `ext`.
3. Improve error handling process where user input is required.
4. Make sure the mounting status of the device is reestablished correctly after the check in case of any errors or interruptions during the checking process.
5. Allow some mechanism for aborting the operation gracefully in the case of unforeseen circumstances.

3.0.376 `n_rescue_install_tools`

Contained in `node-manager/alpine-3/RESCUE/rescue-functions.sh`

Function signature: f29f1eb56f22452e9ae714ba9705886971208f67c053a7a846dc9465346ad370

3.0.377 Function Overview

The `n_rescue_install_tools` function is used to install a set of predefined rescue tools on the system. The function begins by defining an array of package names for the tools to be installed. It updates the system's package index, checks if each package exists in the repository, reports any unavailable packages, and then proceeds to install the valid ones. The function uses logging to display informational, warning, or error messages during execution.

3.0.378 Technical Description

- **Name:** `n_rescue_install_tools`
- **Description:** A Bash function used to check and install a predefined set of rescue tools on a system.
- **Globals:** No global variables are used.
- **Arguments:** None
- **Outputs:** The function will output logs while processing. It reports about updating the package index, about validating the packages, informs about unavailable or valid packages, and finally, about the installation progress.
- **Returns:**
  - 0: If all the valid packages were installed successfully.
  - 1: If there was a failure in updating the package index.
  - 2: If there were no valid packages to install or if there was a failure in installing the packages.
- **Example Usage:** `n_rescue_install_tools`

3.0.379 Quality and Security Recommendations

1. The function should use more error checking to verify uncertainties like network connectivity and permissions.
2. Arguments could be added to allow customization of the array of packages to be installed which will make the function more versatile to use.
3. The function may include a feature to rollback or clean up in case of a failure in package installation.
4. The function should explicitly inform the user in case of no internet access or inability to reach the servers.
5. The function would benefit from more output refinement for better user readability. The logging level could also be added to differentiate between normal, debug, and error logs. This would be beneficial for profiling, testing, and debugging.

3.0.380 `n_rescue_load_modules`

Contained in `node-manager/alpine-3/RESCUE/rescue-functions.sh`

Function signature: c071c704e07cbe0234b5d6d99ae022ec66725b7681f540debd7d4bfce0159c7b

3.0.381 Function overview

The Bash function `n_rescue_load_modules()` is used to load necessary kernel modules in rescue mode. First, it attempts to load ext4 modules, followed by ZFS, and RAID modules. Additionally, it ensures the mdadm tool is available, and if not, it attempts to install it. For each action, it provides logging info and prints a status message to the stderr stream.

3.0.382 Technical description

- **Name:** `n_rescue_load_modules()`
- **Description:** This function tries to load necessary ext4, ZFS and RAID modules. If mdadm tool is not available, it attempts to install it. It logs informational, debug and warning messages, and also gives visual feedback on the stderr stream.
- **Globals:** None
- **Arguments:** None
- **Outputs:** Messages to stderr in the form of visual feedback about the loading/installation status of specified modules and tool.
- **Returns:** `failed` (0 if all loading actions were successful, 1 if not)
- **Example usage:**

```
n_rescue_load_modules
```

3.0.383 Quality and security recommendations

1. Add more detailed comment descriptions for each block of code within the function. This improves readability for future developers.
2. Thoroughly validate all data used by your scripts – even if they seem safe.
3. Use methods that can handle file paths containing spaces, newlines, or other problematic characters correctly.
4. Improve error messages: Rather than just provide a status message, consider providing suggestions for troubleshooting if a module loading attempt fails.
5. Include a timeout for module loading attempts. This can prevent the function from hanging indefinitely when one loading step encounters problems.
6. Set strict mode (`set -euo pipefail`) at the top of your scripts. This will make your script exit if any statement returns a non-true return value. It can help to prevent further problems.

3.0.384 `n_rescue_mount`

Contained in `node-manager/alpine-3/RESCUE/rescue-functions.sh`

Function signature: 74a6713f5fa5c770f001590df58d01a93ffd850581f415521f11619bd756e4d7

3.0.385 Function overview

The `n_rescue_mount` function is designed to execute a rescue mount operation. If root and boot devices are not specified, the function attempts to read them from the IPS configuration. The function also validates the existence of the root and boot devices, handles cases where /mnt is already mounted, mounts the root and boot devices, and prepares the /mnt for chroot.

3.0.386 Technical description

- **Name**: `n_rescue_mount`
- **Description**: Executes a rescue mount operation, reading from IPS configuration if root and boot devices are not specified, validating their existence, handling /mnt mounts, and preparing the /mnt for chroot.
- **Globals**: None
- **Arguments**:
  - `$1` (root_device): The root device to be mounted.
  - `$2` (boot_device): The boot device to be mounted. If specified, a boot mount is executed.
- **Outputs**: Logs of the mounting process, potential error and debug messages.
- **Returns**: The function will return `0` on success, `1` if the specified root or boot device are non-existent or not block devices, and `2` if mounting the root device fails.
- **Example usage**: `n_rescue_mount /dev/sda1 /dev/sda2`

3.0.387 Quality and security recommendations

1. It is recommended to always provide specific root and boot devices when calling the function to decrease reliance on potentially unreliable or outdated configuration files.
2. The function should handle various edge and failure cases such as a failure in the configuration file parsing and mount failures. More comprehensive error handling could improve the function's robustness.
3. Always ensure that the devices specified are valid and intended to prevent accidental data loss.
4. The function could also improve by validating if the specified devices are indeed meant to serve as boot or root. The function currently only checks their existence but more stringent testing could be used to ensure their suitability for booting and root purposes. This can reduce unexpected errors and potential data loss.

3.0.388 `n_rescue_read_disk_config`

Contained in `node-manager/alpine-3/RESCUE/rescue-functions.sh`

Function signature: aa4f296647aa378f9357f49ea0a99ae1e5753e2f0299013d699374a84a46d55c

3.0.389 Function Overview

The bash function `n_rescue_read_disk_config()` is mainly used for reading and outputting the disk configuration from IPS. It attempts to fetch values for five configuration variables (`os_disk`, `boot_device`, `root_device`, `boot_uuid`, `root_uuid`) using the `n_remote_host_variable` function. If no disk configuration is found, the function logs an appropriate debug message and returns 1, else it outputs the configuration and returns 0.

3.0.390 Technical Description

- **Name**: `n_rescue_read_disk_config`
- **Description**: This function reads the disk configuration from an IPS. It fetches values for five variables and checks if any configuration is fetched. If configuration is found, it is outputted and the function signals success. Otherwise, it logs a debug message and signals failure.
- **Globals**: None
- **Arguments**: None
- **Outputs**: If configuration is found, the function outputs variable assignments for `os_disk`, `boot_device`, `root_device`, `boot_uuid`, `root_uuid`.
- **Returns**: `0` if configuration is found; `1` if no configuration is found.
- **Example usage**: `bash source your_script_containing_the_function.sh n_rescue_read_disk_config`

3.0.391 Quality and Security Recommendations

1. For security, it would be better not to suppress the errors (`2>/dev/null`) in the function calls to `n_remote_host_variable`. Errors should be logged or handled properly.
2. Consider validating the inputs to ensure that they are of the expected format and within expected ranges.
3. To make debugging easier, consider adding more detailed logging, for example by logging what value each variable gets or whether each call to `n_remote_host_variable` succeeded or failed.
4. Think about what should happen if getting only some values from `n_remote_host_variable` succeeds and others fail. Currently, if at least one value is fetched successfully, the function returns true.
5. Consider the possibility of using more descriptive function and variable names.

3.0.392 `n_rescue_reinstall_grub`

Contained in `node-manager/alpine-3/RESCUE/rescue-functions.sh`

Function signature: 64ed1794c520e85e541c2e6afe29b904a2027542590188ff1966be7ee70b634c

3.0.393 Function overview

The function `n_rescue_reinstall_grub` is designed to reinstall the GRUB bootloader in a rescue mode. The function first checks if the required OS disk information is present. If not, it aborts the process with an error message. If the disk information is found, the function mounts the required filesystems and then checks if the GRUB packages are installed. If the packages are not found, it installs them. Once the packages are in place, the function runs `grub-install` to install the bootloader. It then verifies the installation followed by updating the GRUB configuration if possible.

3.0.394 Technical description

- Name: `n_rescue_reinstall_grub`
- Description: A Bash function to reinstall the GRUB bootloader.
- Globals: `os_disk`: stores the OS disk configuration.
- Arguments: None.
- Outputs: Log messages and error messages that indicate the progress of GRUB reinstallation.
- Returns: `1` if the OS disk configuration is not found; `2` if the filesystems cannot be mounted; `3` if the GRUB packages cannot be installed or the GRUB installation fails or the GRUB installation cannot be verified. Returns `0` if the GRUB bootloader is reinstalled successfully.
- Example usage: `n_rescue_reinstall_grub`

3.0.395 Quality and security recommendations

1. Use consistent and more detailed messaging for success and failure cases to aid in debuggability.
2. Avoid using `eval` for command substitution due to possible security issues. Consider safer alternatives like parsing.
3. Validate the input values before using them in function.
4. Handle the exception when `grub-mkconfig` is not available neatly by providing a meaningful error message.
5. Use more specific error codes for different types of failure in the reinstallation process.
6. Document the purposes of all global variables used and explain how they fit in the overall program structure since they could potentially affect other operations outside the function scope.
7. Keep security in mind when installing packages, make sure to verify the authenticity of the packages being installed.

3.0.396 `n_rescue_show_help`

Contained in `node-manager/alpine-3/RESCUE/rescue-functions.sh`

Function signature: 998f5c2a4eaaace1f8a01512534be2d6504ae277af782dc54f3bd13e3e353a49

3.0.397 Function overview

The function `n_rescue_show_help()` is used to provide a user interface for accessing a system in rescue mode. This primarily involves logging the display of the help message and outputting a comprehensive text-based guide on how to use the recovery mode's capabilities. Recovery instructions for various scenarios, command details, and general advice are all included in this mode.

3.0.398 Technical description

- **Name:** `n_rescue_show_help()`
- **Description:** This function logs and outputs a detailed, text-based help documentation for use in system rescue mode. It provides guidance for scenarios such as GRUB Repair, Filesystem Repair, Manual Recovery, and others.
- **Globals:** None
- **Arguments:** None
- **Outputs:** The script outputs the rescue mode help directives to the standard error output (`>&2`).
- **Returns:** This function always returns 0, indicating successful execution.
- **Example usage:** To use this function, simply call `n_rescue_show_help` in the shell. No parameters are needed.

3.0.399 Quality and security recommendations

1. Structure the help text in EOF in a more readable and navigable format, using numbered subheadings where necessary.
2. Ensure that all instructions and commands are current and not deprecated.
3. Include more error handling to check whether log messages were successfully sent.
4. Consider breaking up the large text into smaller functions to reduce complexity and improve readability.
5. Always test for unintended consequences before executing commands, especially commands that modify system states or filesystems.

3.0.400 `n_rescue_unmount_recursive`

Contained in `node-manager/alpine-3/RESCUE/rescue-functions.sh`

Function signature: dbc9f7811032cf5f35fbc50a136a70298422f1e6f87faa3008065c8e35170161

3.0.401 Function overview

The `n_rescue_unmount_recursive` function is a bash function specifically designed to unmount a given mount point in a Linux system. The function takes a `mount_point` as input and performs a series of operations to unmount it. It first checks if the `mount_point` is not empty and then verifies if the `mount_point` is mounted before proceeding to unmount it. The function will attempt a recursive unmount if the mount is detected, and in case of failure, it will then try unmounting the specific bind mounts individually. If all these unmount attempts fail, the function log an error message and returns 1.

3.0.402 Technical description

- **Function name**: `n_rescue_unmount_recursive`

- **Description**: The function is designed to unmount a mount point recursively. If the recursive unmount fails, it attempts to unmount the bind mounts individually.
- **Globals**: `n_remote_log`: Used to log information, debug, and error messages.
- **Arguments**:
  - `$1: mount_point`: A string representing the mount point in the filesystem.
- **Outputs**: Debug, information, error messages regarding unmount attempts. Success or failure of unmounting is also outputted.
- **Returns**:
  - `0` if the mount point is successfully unmounted or already unmounted.
  - `1` if the mount_point is undefined or the unmount operation fails.
- **Example usage**:

```
n_rescue_unmount_recursive "/mnt/point_to_unmount"
```

3.0.403 Quality and security recommendations

1. A validation to ensure that the given `mount_point` actually exists in the filesystem can be added, to avoid attempts to unmounting non-existent mount points.
2. Injecting user-controlled input directly into the system command could lead to potential command injection vulnerabilities. Always sanitize the user input before using it in the system commands.
3. Consider using a more robust logging mechanism that will help in understanding the system state and debug issues effectively when necessary.
4. Performance can be improved by executing the unmounting of the bind mounts in parallel.
5. Implement a guard against re-entrant calls.
6. Consider checking and handling other potential errors returned by the `umount` command.

3.0.404 `n_rescue_validate_device`

Contained in `node-manager/alpine-3/RESCUE/rescue-functions.sh`

Function signature: 47e51497a88080e8fb027d3ff373a39138a135f2c063045ae231b2a9f550aa27

3.0.405 Function overview

The function `n_rescue_validate_device()` is a Bash function designed to validate a specified device in the system. It first checks if the device path is empty, and if so, it logs an error and returns a failure status (1). If the device path is not empty, the function checks if the specified path points to a valid block device. If not, it logs another error and returns a failure status. If both checks pass, the function logs a debug message stating that the device has been successfully validated and returns a success status (0).

3.0.406 Technical description

- **Name:** n_rescue_validate_device
- **Description:** This function validates a given device. It checks if the variable "device" is not empty and if it refers to a valid block device in the system.
- **Globals:** None
- **Arguments:**
  - $1: device *(desc: Expected to contain the path to a device in the system)*
- **Outputs:** Error or Debug logs via the "n_remote_log" function, based on whether the validation is successful.
- **Returns:**
  - 1: if the device validation failed (device path is either empty or doesn't reference a block device).
  - 0: if the device path is valid.
- **Example usage:**

```
n_rescue_validate_device "/dev/sda1"
```

3.0.407 Quality and security recommendations

1. Consider adding error handling for when the `n_remote_log` function is not available, as this function relies on it for logging errors and debug information.
2. Conduct input validation apart from emptiness and being a block device, such as checking for permitted characters in the device path to avoid potential command injection.
3. As the function is currently logging all validation messages, it may log sensitive information (like system device path). It is advisable to add an option to control the logging of this potentially sensitive information.
4. To minimize the risk of naming collisions due to the "device" variable being defined with local scope, ensure that variable names follow a well-defined naming pattern across the entire project.

3.0.408 `n_safe_function_runner`

Contained in `node-manager/base/n_safe_function_runner.sh`

Function signature: 47ffa7c7b9700cea52829c4282b7db4c25345d358052ec2abdaf53039bd7ade5

3.0.409 Function overview

The `n_safe_function_runner` function is a safe runner for bash functions with some unique features. Primarily, it runs the specified functions with a timeout mechanism to ensure the function won't be stuck and cause program hanging. The function

also performs some validity checks like the existence of the function, sourcing it from predefined locations if it doesn't exist already. The function makes use of logging if available.

3.0.410 Technical description

Definition Block for Pandoc

- **Name:** n_safe_function_runner
- **Description:** This function acts as a safety wrapper for running bash functions. It provides a timeout mechanism to prevent function from causing a hang. It also checks for the existence of the function, tries to source it from specified locations or logs an error.
- **Globals:**
  - VAR: log_available: determines whether logging is available for use.
- **Arguments:**
  - –timeout: optional, numeric, sets the number of seconds before a function times out.
  - function_name: the name of the function to be run.
  - args: the remaining arguments are passed to the function being run.
- **Outputs:** Echoes an error message if an invalid timeout is provided or if a function name is not provided, or if the function does not exist, or if the function fails or times out. Error messages are directed to STDERR, while log messages are sent to the logging function if available.
- **Returns:** 1 if an error occurred before the function can be run, otherwise the exit code of the function that was run.
- **Example usage:** `n_safe_function_runner --timeout 10 function_name arg1 arg2 arg3`

3.0.411 Quality and Security Recommendations

1. Consider sanitizing function names and arguments. If hostile inputs are injected, it could lead to code injection vulnerabilities.
2. Make sure the path where the function is sourced from is secure and non-writable by non-privileged users to avoid any malicious modifications.
3. Reliable error handling should be in place. The program should not rely solely on the log available or not.
4. Save the error messages and return codes in a more structured way for better downstream debugging or automated analysis.
5. Enhance the timeout mechanism to be more granular (i.e for each individual function in the script) rather than for the whole script execution.

3.0.412 n_select_opensvc_version

Contained in lib/node-functions.d/alpine.d/BUILD/10-build_opensvc.sh

Function signature: b573402d59912fdb59cba6e2ba01581fcd285dab17d0e758b64e5aef21731f67

3.0.413 Function Overview

The n_select_opensvc_version() function is designed to select a specific version of OpenSVC from a local git repository. The function accepts a specific git tag as an argument, which represents the version of OpenSVC to be selected. If no git tag is specified, the function will select the latest version following a semantic versioning (major.minor.patch) pattern in the form of "v..*". The selected version is then checked out from the repository. Environmental variables are set to report the selected version.

3.0.414 Technical Description

- **Name:** n_select_opensvc_version()
- **Description:** Selects a specific tagged version of OpenSVC from a given local git repository or defaults to the latest semantic versioned tag if no specific tag is provided.
- **Globals:** [OPENSVC_GIT_TAG: The selected git tag, OPENSVC_VERSION: The selected version without the leading 'v']
- **Arguments:** [$1: Git tag string representing the desired OpenSVC version]
- **Outputs:** Console messages indicating the function's operations and selected version, logged message of selected version
- **Returns:** Exits with status code 1 if any error occurs (source directory doesn't exist, invalid git repo, requested tag not existing, or checkout failure); exits with status code 0 on successful version checkout
- **Example Usage:** n_select_opensvc_version v2.1.3

3.0.415 Quality and Security Recommendations

1. Be cautious when synchronizing local repositories, which could have potential security implications.
2. Ensure correct validation and error handling for non-existent or invalid repositories. Current version might not correctly handle repositories with no commits.
3. Add more verbose error handling or debugging options to find issues during the checkout process.
4. Avoid printing sensitive information to the console, including the full path of the source directory which can be considered sensitive information.
5. One should consider improving function by allowing users to specify versions using major, minor, and patch arguments separately.

3.0.416 n_set_hostname_and_hosts

Contained in lib/node-functions.d/common.d/n_configure-hostname.sh

Function signature: 8255b960289d7466a858df5b523666f3aadb4ec9119e95a77dc869ed20c0d837

3.0.417 Function overview

This function (n_set_hostname_and_hosts) is used to set the hostname and hosts file on a Linux system. First, it retrieves the required values hostname, domain, and IP address. After validation, it constructs the fully qualified domain name (FQDN) and sets the hostname using hostnamectl command for systemd-based systems or the hostname command for non-systemd systems. This function will also attempt to persist the hostname in /etc/hostname and /etc/hosts file.

3.0.418 Technical description

Here is a technical description of the function:

```
- name: n_set_hostname_and_hosts
- description: This function sets the hostname and hosts file on a Linux system.
- globals: None
- arguments: None
- outputs: Standard output and Log files if n_remote_log is implemented for logging.
- returns: 1 if it fails to get a value from n_remote_host_variable or if either hostnar
- example usage: n_set_hostname_and_hosts
```

3.0.419 Quality and security recommendations

1. Ensure permissions are correctly set for /etc/hostname and /etc/hosts, potentially limiting writing access to these crucial files to root only.
2. Handle edge cases where hostname, domain, or IP data is either missing or incorrect.
3. Use an external configuration management or orchestration system to guarantee the consistency of hostnames across a network.
4. Regularly audit logs (set by n_remote_log) to ensure system's integrity and monitor for failed attempts.
5. Handle failures and exceptions appropriately, letting the function fail gracefully and provide clear, actionable error messages whenever it returns non-zero.
6. Regularly update all the command line tools like hostnamectl, hostname used in the function to the latest stable version to protect against known vulnerabilities.
7. Consider enhancing this function with more advanced features such as IPV6 support.

3.0.420 n_set_state

Contained in node-manager/alpine-3/RESCUE/rescue-functions.sh

Function signature: c6b4895f6502d32d22cb7a411f1b61c86926e8faed0882e18ebc6145ed8bddf4

3.0.421 Function Overview

The function n_set_state() is a Bash function which allows you to set the current state of a process. The function takes a single input, the desired state, and validates it against a list of accepted state values. The function then attempts to set the state variable on a remote host to the specified state. If the state change is successful, the function outputs relevant context regarding this new state, otherwise, it logs an error message, and returns a fail status.

3.0.422 Technical Description

- **Function name**: n_set_state()
- **Description**: Validates a new state against an accepted list of process states, and if valid, attempts to update the state variable on a remote host to this new state. Also provides relevant context based on the new state condition. If the state is not valid or cannot be set, an error message is logged and the function fails.
- **Globals**: None
- **Arguments**: [$1: new_state, The desired state for the process. Should be one of ('PROVISIONING', 'INSTALLING', 'INSTALLED', 'RUNNING', 'FAILED')]
- **Outputs**: Text output to stdout indicating the process status, as well as logging information to the remote host.
- **Returns**: 0 if the state change is successful, 1 if it is not.
- **Example usage**:

```
n_set_state 'INSTALLING'
```

3.0.423 Quality and Security Recommendations

1. The error messages output by the function may reveal implementation details that could be helpful to an attacker. They should be made more generic where possible.
2. The function does not handle cases where the connection to the remote host fails. It would be beneficial to add some form of error handling for these cases.
3. The function does not check for cases where an existing state transition is already in progress, potentially leading to a race condition. Implementing a check for this could improve the function's robustness.

3.0.424 n_setup_build_user

Contained in lib/node-functions.d/alpine.d/BUILD/01-install-build-files.sh

Function signature: 33201ace44e1d96b76fb6d3bcfe80f95d64fa55d4b9702bc58ede4a0356bbf28

3.0.425 Function overview

The `n_setup_build_user` Bash function sets up a user specifically for building APK packages. It checks whether the user exists, creates one if they don't, ensures the user's home directory exists with the correct permissions, configures the user's group membership, checks for (and generates, if necessary) a signing key for ABUILD, and sets up permissions for the package directory. If the function is successful, it will also log its completion.

3.0.426 Technical description

- **Name**: `n_setup_build_user`
- **Description**: This function creates and configures a build user for APK package creation.
- **Globals**:
  - `build_user`: The name of the user being created and configured for package creation.
  - `packages_dir`: The directory that will contain the built packages.
- **Arguments**: None
- **Outputs**: Informational messages on the console informing of the function's progress. Error messages will be displayed if user creation or signature key generation fail.
- **Returns**: `0` on successful completion, `1` if user creation or signature key generation fail.
- **Example usage**:

```
source path/to/function.sh
n_setup_build_user
```

3.0.427 Quality and security recommendations

1. It would be better to handle errors more robustly - consider sending error messages to stderr rather than stdout.
2. For improved security, consider adding checks to verify that the current user has enough permissions to create a new system user and modify file system permissions.
3. Utilize secure bins for creating or modifying users to limit potential for malicious code introduction.
4. The function should also handle edge cases, such as the users and groups already existing, or a directory being a file.

3.0.428 n_setup_ntp

Contained in lib/node-functions.d/alpine.d/alpine-lib-functions.sh

Function signature: 6c843755e4136809f8866301e855b9467def2e6b10fbe51d27ac75b8bc878f1b

3.0.429 Function Overview

The n_setup_ntp function attains time synchronization configuration for a given server. It first retrieves the NTP server from a cluster configuration. If an NTP server was not set in the cluster config, it uses a default server, 'pool.ntp.org'. The function then creates an NTPD configuration file. If it fails to create this file, it logs an error message and returns a failure message (1). If it succeeds in creating the configuration file, it logs a success message and returns a success message (0).

3.0.430 Technical Description

Function Name: n_setup_ntp

Description: A configuration function that sets up NTP (Network Time Protocol) time synchronization, using server settings from a cluster configuration or a default.

Globals:configures NTPD_OPTS in /etc/conf.d/ntpd

Arguments: None

Outputs: Depending on the successful execution of the function, the function logs either a success message or an error message onto the console.

Returns: Returns 1 in case of a failure to create the ntpd configuration. Otherwise it returns 0 (indicating success).

Example Usage:

```
n_setup_ntp
```

3.0.431 Quality and Security Recommendations

1. Verify the ntpd config file before using it in your configuration file. This checking increases the reliability of your synchronization.
2. Reasonable default servers such as pool.ntp.org should be used when the NTP servers are not configured.
3. In depth validation and error handling should be implemented to avoid synchronization failures. This might include testing and validating user inputs and checking the availability and response of sync servers.
4. Logging should be detailed enough to debug problems in case of failure. It should include timestamps and other identifiers essential for the debugging.

5. Ensure secure communications with NTP servers, to prevent potential attacks altering time information. Configuration should prefer using secure NTP, as well as ensuring secure key management and enforcement of authentication.

3.0.432 n_show_vlan_interfaces

Contained in lib/host-scripts.d/common.d/n_network_functions.sh

Function signature: 7be6dc67621a499ff331c0f30e602bda79e77117b88d6451d7e8f309ae8e9dbc

3.0.433 Function Overview

The function n_show_vlan_interfaces outputs information about the VLAN interfaces of a device. It begins by printing a heading "VLAN Interfaces:" and an underline. Next, it uses the ip command to iterate over each network interface that's using the VLAN protocol. It extracts and displays the following information for each qualifying interface: the interface's name, VLAN ID, IPv4 address, operational state, and the Maximum Transmission Unit (MTU) size.

3.0.434 Technical Description

```
- Name: n_show_vlan_interfaces
- Description: Outputs a report of VLAN interface information.
- Globals: None
- Arguments:
  - No arguments are used in this function.
- Outputs:
  - A formatted list of VLAN interfaces is output to STDOUT,
 ↪  including their interface names, VLAN IDs, IP addresses,
 ↪  operational states, and MTU sizes.
- Returns:
  - Returns nothing as the function does not explicitly handle
 ↪  return values.
- Example Usage:
  Call the function simply with `n_show_vlan_interfaces`.
```

3.0.435 Quality and Security Recommendations

1. Introduce input validation and error checking mechanisms to enhance robustness. Also, consider print error messages to STDERR instead of STDOUT.
2. Use local -r for variables that are not meant to be changed to enforce immutability and improve code safety.
3. Enclose all variable references in double quotes to prevent word-splitting and pathname expansion.

4. It's advisable to handle any potential errors that may occur when using `cat` command to read from system files. If these files cannot be read, the error messages should be appropriately handled.
5. Collect the script's dependencies (e.g., `ip`, `awk`, `cut`, `grep`) at the start of the script and notify the user if any are missing.
6. Consider hardening the function by employing a stricter globbing or regex match to ensure it only acts upon valid interface identifiers.
7. Given it's a public function, it's advisable to implement a general STDERR logging mechanism. If needed, switch between verbosity levels.

3.0.436 `n_start_base_services`

Contained in `lib/node-functions.d/alpine.d/alpine-lib-functions.sh`

Function signature: 17600a766548ab152a931cfdeeef33f47b55a8e9250cee5cc92b178b14f87cf5

3.0.437 Function Overview

The `n_start_base_services` is a Bash function within a larger script context and it is designed to initiate the base system services for a computer system. The function first logs that it is going to start the system services. Then, it specifies the names of the services to be initiated in an array. It checks each service to see if it is already running and if not, it starts the service while logging its activities. In case of failure to start a service, a warning message is generated.

3.0.438 Technical Description

- **Name:** `n_start_base_services`
- **Description:** This function starts base system services such as hardware drivers, kernel modules, filesystem check, root filesystem, local filesystems, and system hostname. It makes sure these services are up and running, if not, it starts them and logs its actions.
- **Globals:** None
- **Arguments:** None
- **Outputs:** Logs the action of starting the services, and any warning messages if a service fails to start.
- **Returns:** The function will always return `0` implying that it has finished its operation, regardless of whether all services started successfully or not.
- **Example Usage:** To use the function, you generally just need to call it in your script like: `n_start_base_services`.

3.0.439 Quality and Security Recommendations

1. Always ensure that this function runs with the appropriate permissions required to start system services. If not, some services may fail to start, even though the software thinks they were started properly.
2. Consider adding error handling functionality to react to the case when a service fails to start. This could involve trying to start the service again, or possibly alerting the user or administrator.
3. Implement input validation. Although this function does not receive input parameters, it is always good practice if it does in the future.
4. Ensure that logging information does not reveal sensitive data or expose the system to any security risks. Moreover, log files should be periodically reviewed to ensure system services are running as expected.

3.0.440 `n_start_libvirt`

Contained in `node-manager/alpine-3/TCH/KVM/install-kvm.sh`

Function signature: a58e526839114218b53cb6a74feeacd86598363892963630e2fbebc22ba6a5c8

3.0.441 1. Function overview

The `n_start_libvirt` is a Bash function that aims to start the services for “libvirtd” and “virtlogd”. Each of these services has a distinct role in the management of virtualization platforms. The `libvirtd` service is responsible for managing platforms like the Xen virtualization manager, while `virtlogd` acts as a logging daemon for `libvirt`.

3.0.442 2. Technical description

name ‘n_start_libvirt’

description This function starts the libvirtd and virtlogd services using the rc-service command.

globals None

arguments None

outputs This function does not explicitly output any values

returns This function does not return any specific value. If succesful, the services libvirtd and virtlogd start running.

example usage

```
n_start_libvirt
```

3.0.443 3. Quality and security recommendations

1. Ensure the existence of `libvirtd` and `virtlogd` services before calling the function in order to avoid runtime errors.
2. Add error handling to deal with possible scenarios where the start of the services could fail.
3. Include service status verification commands in the function to confirm that the services have started correctly.
4. For security purposes, ensure that only authorized users or services can start or stop these services.
5. Add comments to the code for clarity and maintainability.

3.0.444 `n_start_modloop`

Contained in `node-manager/alpine-3/alpine-lib-functions.sh`

Function signature: eca40ab59936c17208c1cf1d3f65ca64a52949b467fd13d3640790592b8f789b

3.0.445 1. Function Overview

This function, `n_start_modloop`, is designed to ensure that the `modloop` service is up and running on a system serviced by OpenRC. It first checks the status of `modloop` service. If the service is not currently running, the function starts the service.

3.0.446 2. Technical Description

- **Name**: n_start_modloop
- **Description**: This function is used in the setting of OpenRC to manage the `modloop` service. It firstly checks the status of the service and if the service is not active, it starts the service.
- **Globals**:
  - `rc-service`: This global variable is an OpenRC command. It is responsible for controlling OpenRC services.
- **Arguments**: None.
- **Outputs**: Depending on the status of the `modloop` service, it can output the status message or the result of the attempt to start the service.
- **Returns**: Returns the exit code of the `rc-service modloop start` command if the `modloop` service was not running. If the service was already active, it returns the exit code of the `rc-service modloop status` command.
- **Example usage**:

```
n_start_modloop
```

3.0.447 3. Quality and Security Recommendations

1. Implement error handling: This function does not handle the case when the `rc-service` command fails to execute, or if the `modloop` service fails to start. Error handling should be added for these scenarios.
2. Provide feedback: It would be useful to provide feedback to the user whether the service was already running, or if it was started by the function.
3. Check for root permissions: Since the `rc-service` command requires root permissions to start services, the function should check that it has the necessary permissions before attempting to alter service status.
4. Use absolute paths for commands: To avoid potential issues with PATH environment variable, using absolute paths for system commands could be safer.
5. Validate results: After starting a service, it would be good to validate that it started correctly and is running, instead of assuming that the start command succeeded.

3.0.448 `n_storage_auto_configure`

Contained in `node-manager/base/n_network-storage-functions.sh`

Function signature: 55c01e597e7f0411d3ad5dbe517c5de657ce3823e117d7e2336947a6619eb87f

3.0.449 Function Overview

The `n_storage_auto_configure` function is used primarily in a network storage environment to automate configuration of network interfaces and IP allocations from an IP Scope Service (IPS). The function takes two optional parameters: a storage index, and an override network interface name.

3.0.450 Technical Description

- **Name**: `n_storage_auto_configure`
- **Description**: The function automates the configuration of network interfaces and IP allocations for a network storage setup by specifying a storage index and an optional override interface. If an interface override is provided, it saves it for persistence. When no override is given, it selects an interface based on the storage index. Once an interface is selected, it brings up the interface if it's not already up, gets the allocation of IP from the IP Scope Service (IPS), parses and configures the vlan and interface using the obtained IPs.
- **Globals**: None.
- **Arguments**:
  - `$1`: Storage Index (default 0); Index to use when selecting a network interface or saving override.

  - $2: Override Interface (optional); The name of the network interface to use instead of selecting based on index.

- **Outputs**: Logs messages about the allocation, configuration, and errors if any.
- **Returns**: Returns 0 on successful allocation and configuration, otherwise 1.
- **Example usage**:

```
n_storage_auto_configure 1 ens160
```

3.0.451 Quality and Security Recommendations

1. Validity of the `storage_index` and `override_iface` variables should be checked before use.
2. Check and handle any possible exceptions or errors during the execution of the ip and vlan commands.
3. Increase the general robustness of the script through better error handling and confirmation of successful operations.
4. Consider securely logging the activities and errors for future reference and auditing purposes.
5. Wherever possible, limit the permissions of the function as much as possible so that it can only perform the necessary tasks.

3.0.452 `n_storage_network_setup`

Contained in `node-manager/base/n_storage-functions.sh`

Function signature: b9d955ab898629611ccbb3c79adf2cb697c037821b05b564e6e5b61ababbb529

3.0.453 Function overview

The `n_storage_network_setup()` is a Bash function designed for setting up the network for storage in a system. It uses the physical interface and storage index as parameters. It obtains the MAC for ID, requests an IP from IPS, does an error check and sets up the VLAN. It adds the IP and stores the result in host variables. The function also tests the gateway through a ping.

3.0.454 Technical description

- **name:** n_storage_network_setup()
- **description:** Function to set up a network for storage in a system using a physical interface and storage index.
- **globals:** [None]
- **arguments:**
  - $1: The physical interface for network setup
  - $2: The storage index number (defaults to `0` if not specified)

- **outputs:** Logs for network setup progress and status
- **returns:** `0` if successful, `1` if an error occurred
- **example usage:**

```
n_storage_network_setup eth0 1
```

3.0.455 Quality and security recommendations

1. Implement input validation to ensure that the arguments provided to the function are in the correct format and within the expected range.
2. Handle potential issues, such as unavailability or failure of the MAC address reading, IP allocation, VLAN creation, or interface IPs adding.
3. Ensure that errors and failures provide clear, precise, and useful logging messages to help troubleshooting.
4. Always use the latest secure versions of the libraries and tools used within this function.
5. Consider edge cases, like if the physical interface is not available, or the IP allocation returns unexpected format.
6. Utilize a secure method for storing and retrieving host variables.
7. Secure any handling of data or communication that could be susceptible to injection attacks or eavesdropping.

Also, always follow good security practices while implementing any networking function, including the principle of least privilege (PoLP), segregation of duties (SoD), and ensuring strong access controls. Regular audits and updates are also essential to maintain the security of the function.

3.0.456 `n_storage_provision`

Contained in `lib/node-functions.d/common.d/n_network-storage-functions.sh`

Function signature: ba69ec0bbe59f70deafa36ae19024de2b96727c3314f0a7a5f55a78225961458

3.0.457 Function Overview

The provided Bash function `n_storage_provision()` is designed to manage the provisioning of storage networks on a specific system. The primary tasks it executes include loading network modules, checking the configuration for storage count, detecting the network interfaces (with simple interface detection), configuring each network, and logging information throughout the entire process.

3.0.458 Technical Description

- `name: n_storage_provision()`
- `description:` A Bash script function to handle the provisioning of storage networks on a system.

- `globals:` None.
- `arguments:` None.
- `outputs:` Displays informative logs on the console about the progress of the provisioning process. These logs display error messages when configuration or allocation fails, as well as detailed information on storage allocation, configuration, and completion.
- `returns:` Returns 1 if any initial configurations (like getting the storage count) or if there are no available interfaces for storage fail. Returns 0 upon successful completion of the full process.
- `example usage:`

```
n_storage_provision()
```

3.0.459 Quality and Security Recommendations

1. Consider better handling for when the function fails to find enough interfaces or fails to complete other steps. Right now, it may continue to further steps even if previous steps failed, depending on the specific failure.
2. Validate inputs and outputs of invoked sub-procedures to minimize the chance of erroneous data propagation and ensure data integrity.
3. Evaluate necessity of `sleep 2` delays. These may not be needed, and if they are necessary for some reason (like waiting for network modules to load), consider implementing a more robust solution.
4. Ensure more descriptive logging detailing the actions being undertaken to provide administrators additional clarity on potential points of error.
5. Check the execution permissions for scripts invoking the function to prevent unauthorized running. Only specific, authorized roles should be able to manage storage networks.
6. Regularly update and maintain the function to address changing technology or security requirements. Always follow best practices for scripting to minimize the risk of potential security vulnerabilities.

3.0.460 `n_storage_select_interface`

Contained in `lib/node-functions.d/common.d/n_network-storage-functions.sh`

Function signature: abd5da8f8577fd3d700bf0bbc0c7294bcd46e47dad9ac1620ceaf33ad61633ba

3.0.461 Function overview

This Bash function, `n_storage_select_interface`, handles the processes of selecting a network interface for a storage network. The function first checks if there's a preselected and persisted network interface. If the selected interface still exists in system, the function will return it; otherwise, it logs a warning. Then, the function tries to find an unconfigured interface to use. If no such interface is found, the function logs an error and

returns 1. If an unconfigured interface is found, the function persists this as the selected interface, logs the selection, returns the selected interface.

3.0.462 Technical description

- **Name**: `n_storage_select_interface`
- **Description**: This function is responsible for selecting a network interface for a storage network. It either reuses a persisted selection or selects a new unconfigured interface.
- **Globals**:
  - `storage_index`: The index designating a specific storage network.
- **Arguments**:
  - `$1`: The index of the storage network. Defaults to 0 if no argument is provided.
- **Outputs**:
  - The name of the selected network interface.
  - Logs errors or warnings as needed.
- **Returns**:
  - Returns 0 if a suitable network interface is found and selected.
  - Returns 1 in case of no suitable interface is found.
- **Example Usage**:

```
interface=$(n_storage_select_interface 2)
echo "Selected interface for storage network 2 is $interface"
```

3.0.463 Quality and security recommendations

1. It's recommended to include error handling to ensure the provided storage index is of the correct type (integer) and within the correct range.
2. Look out for potential race conditions or clashes with other processes trying to configure the same network interface at a similar time. Consider using locks or mutexes where applicable.
3. The function is using several other functions (n_remote_host_variable, n_network_find_unconfigured_interface, n_remote_log) that are not described here. Ensure that those functions are carefully reviewed for security and robustness.
4. Network interfaces can have complex states. It's recommended to use a more reliable method to verify if a network interface still "exists" other than just checking if its directory in /sys/class/net exists.
5. Consider the privacy implications of logging network interface information, particularly in environments where this information might be sensitive. Ensure compliance with any relevant regulations or guidelines.

3.0.464 n_storage_unconfigure

Contained in `lib/node-functions.d/common.d/n_network-storage-functions.sh`

Function signature: 9f17ea5a9d969a3467b99f10055fa6ba722e863beb2a4db8203979e2a56577e2

3.0.465 Function Overview

The `n_storage_unconfigure()` function is used to unconfigure a storage index in a network, removing DNS entries and VLAN interfaces. It also clears host variables connected to the specified storage index.

This function is primarily useful in large, complex networks where specific storage nodes need to be cleanly unconfigured, ensuring that there are no remaining hooks in network setup or DNS configurations.

3.0.466 Technical Description

- **Name:** `n_storage_unconfigure()`
- **Description:** This function unconfigures a specific storage index in a network. It clears host variables, removes DNS entries if an IP is specified, and also removes the VLAN interface associated.
- **Globals:** None
- **Arguments:**
  - `$1: Storage Index` - The storage index you wish to unconfigure. Default is 0.
- **Outputs:**
  - Outputs log messages to the console logging the actions performed, such as DNS removals.
- **Returns:** It returns 0 after successful execution.
- **Example Usage:**

```
n_storage_unconfigure 1  # Unconfigures the Storage Network
 ↪  with index 1
```

3.0.467 Quality and Security Recommendations

1. Input Validation: Ensure that input parameters are validated to avoid unexpected issues or security vulnerabilities. The storage index should be a valid integer.
2. Error Handling: Better error handling can be implemented during Removal of DNS and VLAN interface to ensure that any errors during these processes are caught and properly handled.

3. Logging: Logging can be made more detailed or adjustable by level to help debugging and usage understanding. It can provide vital clues if there is failure.
4. Code Comments: While the function is quite clear, adding comments for the more complex parts of the function will improve readability and maintainability.
5. Secure deletion: When removing DNS entries or network settings, ensure the deletion is secure and cannot be easily snooped on or interfered with.

3.0.468 n_storage_validate_jumbo_frames

Contained in `lib/host-scripts.d/common.d/n_storage-functions.sh`

Function signature: 88bd0071a499bbd8569aa662f5b5a8dc21749e5558c21d25b8da39efca395556

3.0.469 Function Overview

The function `n_storage_validate_jumbo_frames()` checks for the validity of jumbo frames in a network. It accepts three arguments where the third argument is optional and its default value is set to 9000. The function calculates the size of the ping packet, tests the ability of jumbo frames to reach the target IP with the calculated packet size and logs the activity in both success and fail scenarios. It returns `0` if the test was successful (indicating the validity of jumbo frames), and `1` if it was not successful.

3.0.470 Technical Description

- Name: `n_storage_validate_jumbo_frames`
- Description: Function to validate jumbo frames in a network.
- Globals: None
- Arguments:
  - `$1` (vlan_iface): The Vlan Interface in question.
  - `$2` (target_ip): The Target IP where the jumbo frames are being sent.
  - `$3` (expected_mtu): The Expected Maximum Transmission Unit. It is optional, defaulting to 9000.
- Outputs: Logs to `n_remote_log`
- Returns: `0` if jumbo frames are validated successfully, `1` if the validation fails.
- Example usage: `n_storage_validate_jumbo_frames eth0 192.168.0.1 8900`

3.0.471 Quality and Security Recommendations

1. Ensure that the network firewall rules do not block the ICMP protocol used for pinging.
2. Verify that the Vlan interface and target IP passed as parameters are valid and reachable.
3. Exception handling can be improved by adding checks for the argument values.

4. Ensure that the error message logged is informative, indicating the possible reasons for failure.
5. Rather than suppressing shell command errors (&>/dev/null), consider handling them to provide more feedback on what might have gone wrong.
6. To protect against command injection, before executing commands with arguments supplied by user, sanitize the inputs.
7. Consider encapsulating the logic of calculating packet size into a separate function to make the code cleaner and more modular.

3.0.472 `n_url_encode`

Contained in `lib/host-scripts.d/pre-load.sh`

Function signature: 4e21e5663f425634af5a4baa7396afcc1884c8ecdeb58e9f5ffd146304fd834a

3.0.473 Function Overview

This function, `n_url_encode()`, is used to encode a provided string in the URL encoding standard. Each non-alphanumeric character in the input string is replaced with its hexadecimal representation prefixed by '%'. Alphanumeric characters, as well as '.', '~', '_', and'-', are left as-is. Spaces are also replaced by'%20'.

3.0.474 Technical Description

- **Name**:
  - `n_url_encode()`
- **Description**:
  - This function encodes a provided string into URL encoding format by converting each non-alphanumeric character into a hexadecimal representation prefixed by '%'.
- **Globals**:
  - `(LC_ALL=C)`: This changes the locale to C to ensure predictable character classification.
- **Arguments**:
  - `($1)`: This is the string input that will be URL encoded.
- **Outputs**:
  - There is no explicit output. The result is printed to the standard output.
- **Returns**:
  - The function returns a new URL encoded string.
- **Example Usage**:

```
n_url_encode "Hello, World!"
# Outputs: Hello%2C%20World%21
```

3.0.475 Quality and Security Recommendations

1. Input validation should be implemented for the function argument. This ensures that the function input is a string before attempting to perform the URL encoding operation.
2. For better readability and maintainability, consider using meaningful variable names instead of single letters.
3. Each special character used should have a clear comment indicating why it's being included in the URL encoding process.
4. ShellCheck warnings should not be ignored unless absolutely necessary, as they might indicate potential bugs or security vulnerabilities. Issue SC2039 is being ignored here, which relates to the use of undocumented/bin/sh features, and should be addressed appropriately.
5. Try to avoid using global variables where possible, as this can lead to unwanted side-effects if they are modified elsewhere in the script.
6. Although this function can be used in any shell script, care should be taken not to use it on data that is already encoded, as it may lead to double encoding issues. Plan your code to avoid such situations.

3.0.476 `n_vlan_create`

Contained in `lib/node-functions.d/common.d/n_network-functions.sh`

Function signature: ef56d7f556ee4dc1eff461d4554ebcd6fa4417a20a048f5a0cb407cd62b05eb2

3.0.477 Function Overview

The `n_vlan_create` function is used to create a VLAN (Virtual Local Area Network) on a physical network interface. It takes three arguments - the physical interface to create the VLAN on, the VLAN ID, and optionally, the MTU (Maximum Transmission Unit). The function first validates these arguments, then checks the existence of the physical interface. If a VLAN already exists on that interface with the same ID, it's deleted and a new one is created. An MTU check is conducted and if the requested MTU is larger than the physical interface's current MTU, it's updated. If there are any issues during these steps, appropriate error or warning logs are generated. Finally, the newly created VLAN is enabled.

3.0.478 Technical Description

- **Function Name**: `n_vlan_create`

- **Description**: This function creates a virtual local area network (VLAN) on a specified physical network interface, with a specified VLAN ID and Maximum Transmission Unit (MTU).
- **Globals**: `phys_iface`, `vlan_id`, `mtu`
- **Arguments**:
  - `$1`: The physical network interface on which to create the VLAN.
  - `$2`: The ID of the VLAN to be created.
  - `$3`: The Maximum Transmission Unit (MTU) size. This argument is optional and defaults to 1500 if not provided.
- **Outputs**: The function will echo usage instructions if the required arguments are not supplied. It will also log error and warning messages to console during execution.
- **Returns**: `1` if an error occurs; `0` if the function successfully created and enabled the VLAN.
- **Example Usage**: `n_vlan_create eth0 100 1600`. This command creates a new VLAN with ID 100 on `eth0` with an MTU of 1600.

3.0.479 Quality And Security Recommendations

1. Input validation: Currently, there is no check to ascertain whether the provided VLAN ID falls within the accepted range (1-4094). Implement this check to prevent invalid VLAN IDs.
2. Error handling: The function should handle errors more gracefully. Rather than continuing with process even after a failure, reconsider exit after a serious error.
3. Usage of sudo: All `ip link` commands require root privileges. So, either run the function with `sudo` or use the command `sudo` within function to prevent permission issues.
4. Secure logging: Use secure logging mechanisms like `syslog` for error or output logging, rather than directly outputting to console. This can provide better access control and auditing capabilities.
5. Harden the use of `sleep 1`: Usage of sleep without any condition might lead to unnecessary process delay or blocking. Instead, implement a wait mechanism with verification on `ip link delete` operation.

3.0.480 `n_vm_create`

Contained in `lib/node-functions.d/common.d/n_vm-functions.sh`

Function signature: a35e117904d42b745641766476c0f3afad0f22d188ee42b146198071091138d6

3.0.481 Function Overview

This is a bash script function that creates a virtual machine. The function follows the following steps:

- It verifies the parameters
- Fetches the VM configuration
- Parses the returned configuration
- Validates the required fields
- Verifies the provision method
- Builds the `virt-install` command
- Executes `virt-install`.

Throughout the process, the function logs information about the steps and any errors encountered.

3.0.482 Technical Description

Function Name: `n_vm_create`

Description: A bash function for creating a Virtual Machine.

Globals:

- `n_remote_log`: Function for logging messages.

Arguments:

- `$1 (vm_identifier)`: Unique identifier for the VM.
- `$2 (override_title)`: Optional parameter that overrides VM's title from the fetched configuration.
- `$3 (override_description)`: Optional parameter that overrides VM's description from the fetched configuration.

Outputs:

- Outputs information during the process including errors and logs them using the `n_remote_log` function.

Returns:

- `1`: If required argument is missing or empty.
- `2`: When there is a failed or empty fetch configuration.
- `3`: If a required field is missing in the configuration.
- `4`: If a non-supported provision method is chosen.
- `5`: When `virt-install` command fails.
- `0`: On successful creation of VM.

Example Usage:

```
n_vm_create "vm1" "Test VM" "This is a test VM"
```

3.0.483 Quality and Security Recommendations

1. Consider adding input validation for the VM configuration parameters.

2. Error and exception handling can be further improved by capturing and logging more specific error messages.
3. Variables such as `provision_method` can be set to be read-only to prevent unnecessary modifications.
4. Apply shellcheck and other linter tools to ensure the syntax and best practices are adhered.
5. Ensure sensitive logs and outputs have proper confidentiality and integrity controls.
6. Add more comprehensive tests to verify the functionality and identify potential errors.

3.0.484 n_vm_destroy

Contained in `lib/node-functions.d/common.d/n_vm-functions.sh`

Function signature: a8c714998f2e749d781b78868c42ef6a2fb92f8c4a57c3aa4e8fbff21b101a51

3.0.485 Function overview

The `n_vm_destroy` function is designed to stop and undefine a virtual machine on the host system. The function accepts a single argument, which is the identifier of the virtual machine to be destroyed. The function uses the `virsh` command to perform the task and logs every step it performs.

3.0.486 Technical description

```
name: `n_vm_destroy`
description: Stops and undefines a virtual machine on the host
 ↪  system using its identifier.
globals: [ `n_remote_log`: The function used to log information
 ↪  and errors. ]
arguments:
[
    `$1: vm_identifier`: The identifier of the virtual machine to
 ↪  destroy.
]
outputs:
- Logs every step of the operation, success messages for
 ↪  successful destruction and error messages for missing
 ↪  vm_identifier parameter, empty vm_identifier or failure of the
 ↪  undefinition of the VM.
returns:
- `1`: If the vm_identifier is missing or empty, or if the vm
 ↪  undefinition failed.
- `0`: If the VM was successfully destroyed.
example usage: `n_vm_destroy example_vm`
```

3.0.487 Quality and security recommendations

1. Implement proper validation for the `vm_identifier` input.
2. Prevent possible code injection through the `vm_identifier` by sanitizing the input.
3. Avoid suppressing errors completely to ensure that significant errors do not go unnoticed.
4. Use personalized return codes to differentiate between the different errors that may arise from the function.
5. For security, consider enforcing authentication measures when attempting to destroy a VM.
6. Log any non-normal operations or behaviors to ensure traceability of actions.
7. Make sure the function stops its execution as soon as a critical error occurs to avoid causing potential harm to the system.

3.0.488 `n_vm_pause`

Contained in `lib/node-functions.d/common.d/n_vm-functions.sh`

Function signature: b621e70a2c9f8795eab227781fe63eeaba6b902bf6ed91e46850aba755e34d77

3.0.489 Function overview

The `n_vm_pause()` function is built to pause a virtual machine using the virsh command, with the given VM identifier. It consists of validation for the VM identifier and logging for operations performed. The function will log if the function fails to pause the VM or the VM identifier is missing or empty. The function will also log once the VM is successfully paused.

3.0.490 Technical description

- **name**: `n_vm_pause()`
- **description**: Pauses a virtual machine using the virsh command.
- **globals**: None
- **arguments**:
  - `$1: vm_identifier`: Identifier for the virtual machine to be paused.
- **outputs**: Information logs on the operation executed and any potential errors.
- **returns**:
  - `0` if the VM is successfully paused.
  - `1` if the function failed to pause the VM or if the VM identifier is missing or empty.
- **example usage**: `n_vm_pause <vm_identifier>`

3.0.491 Quality and security recommendations

1. Always ensure that the VM identifier is correctly provided to avoid logging and execution errors.
2. Strict checking can be added for VM identifiers if they need to follow a certain format or pattern.
3. Handling for edge cases can be improved. For example, when a VM that does not exist is provided as the identifier.
4. Implement additional error logging and handling during the execution of virsh command.
5. Consider validating the VM's status before attempting to pause, such as whether it is already paused or stopped.
6. Make sure only those with proper privileges can execute this function to avoid any potential security risks.

3.0.492 `n_vm_start`

Contained in `lib/node-functions.d/common.d/n_vm-functions.sh`

Function signature: b683ab5fc4bb2f7fc09ffbb062eb290f855679b0b2239f291e64e148144aea1d

3.0.493 Function overview

The function `n_vm_start` is designed to initiate a virtual machine through an identifier provided as an argument. The function first validates that an identifier is provided and is not empty, logging an error message otherwise. After this, the function initiates the virtual machine using the `virsh start` command. Depending on the response from this command, the function will log either a success message, or an error message with the response from the `virsh start` command.

3.0.494 Technical description

- **Name:** `n_vm_start`
- **Description:** The function starts a virtual machine by taking the machine's identifier as an argument.
- **Globals:** None.
- **Arguments:**
  - `$1`: `vm_identifier`, the identifier of virtual machine to be started.
- **Outputs:** Logs messages showing either successful start of the virtual machine or error messages if the operation fails.
- **Returns:** Returns 1 (error) if there aren't exactly 1 arguments passed, if the VM identifier is not provided, or if the virsh command fails. Returns 0 (success) if the VM starts successfully.

- **Example Usage:**

```
n_vm_start "centos7-vm1"
```

3.0.495 Quality and security recommendations

1. Ensure that input validation is robust. In particular, guard against injection attacks where a rogue VM identifier might include unexpected characters that could execute harmful shell commands.
2. Use of more specific error codes can help identify type of error encountered.
3. Proper logging of steps would make debugging easier.
4. Exception handling could be improved. Instead of just returning 1 on failure, the function could re-attempt the operation a certain number of times before giving up.
5. Ensure least privilege principle is followed, i.e, the program using this function should only have the bare minimum privileges required to function.
6. Protect the log files that are written by this function. They might contain sensitive information that should not be accessible to just anyone with access to the file system.

3.0.496 `n_vm_stop`

Contained in `lib/node-functions.d/common.d/n_vm-functions.sh`

Function signature: 73ac108ba536601d31e01eb206690b16390c6b76ff43b988d58ceb2b56c9b3db

3.0.497 Function overview

The `n_vm_stop` function is used to shutdown or forcefully destroy a specified Virtual Machine (VM). It starts by validating the provided `vm_identifier` parameter, which is used to identify the VM to be operated on. If the second optional parameter 'force' is provided, the function will forcefully destroy the VM by running the `virsh destroy` command. If not, it will attempt to gracefully shutdown the VM using the `virsh shutdown` command. The function reports all its activities by making calls to the `n_remote_log` function.

3.0.498 Technical description

- name: n_vm_stop
- description: Stops or destroys a specified Virtual Machine (VM).
- globals: None
- arguments:
  - `$1: vm_identifier` (required): Identifier of the VM to be stopped.
  - `$2: force` (optional): If "force" is provided, the VM will be forcefully destroyed. Otherwise, a graceful shutdown is attempted.

- outputs: Logs all operations to a remote system log by making calls to the `n_remote_log` function.
- returns: Returns `0` if the operation was successful, and `1` otherwise.
- example usage: `n_vm_stop "VM1" "force"`

3.0.499 Quality and security recommendations

1. The function should make sure to handle situations when the virsh command is not available on the system. Perhaps, checking the existence of virsh before executing would be a good strategy to reduce possible errors.
2. Currently, the function does not handle scenarios where the specified VM does not exist. Adding in this validation can improve the robustness of the function.
3. Ensure that permission checks or necessary sanitization are in place for the input to this function to prevent any potential security issues.
4. The function only logs errors and successes. It might be helpful to log more detailed information to assist in debugging and maintenance.
5. The function currently does not handle a situation where the force parameter is set to something different from 'force'. Making the force parameter checking case insensitive, could improve usability.

3.0.500 `n_vm_unpause`

Contained in `lib/node-functions.d/common.d/n_vm-functions.sh`

Function signature: 8f700d0561d049973f19839565f481ed9109fe6183eff3c49522da0490b84a98

3.0.501 Function Overview

The function `n_vm_unpause()` is designed to resume or unpause a virtual machine (VM) that has been previously paused. It takes a single argument `vm_identifier`, which is an identifier for the VM to be resumed. The function validates the received argument and logs the operation. It then executes the `virsh` command with appropriate options to resume the VM. If the VM resumes successfully, it logs this operation as a success, otherwise, it logs the error message and returns 1.

3.0.502 Technical Description

- **Name**: `n_vm_unpause`
- **Description**: Unpauses a previously paused virtual machine.
- **Globals**: None
- **Arguments**:
  - `$1: vm_identifier` - Identifier for the VM to be resumed.

- **Outputs**:
  - Logs an error message if `vm_identifier` parameter is missing or empty.
  - Logs information about the status of the VM unpausing operation.
- **Returns**: 0 if successful; 1 if unsuccessful.
- **Example usage**:

```
n_vm_unpause my_vm
```

3.0.503 Quality and Security Recommendations

1. *Input sanitation*: Always sanitize and validate your inputs. In this case, `vm_identifier` should be thoroughly checked to prevent potential command injection.
2. *Use of quotes*: Ensure to always quote your variables to prevent word-splitting and pathname expansion.
3. *Error handling*: Consider advanced error handling mechanisms, e.g., catching and dealing with any exception raised by `virsh`.
4. *Documentation*: Maintain a documented standard for the log messages to ensure consistency.
5. *Testing*: Write unit tests to make sure your function behaves as expected in all situations. This also aids in spotting a defect or issue earlier.

3.0.504 package

Contained in `node-manager/alpine-3/TCH/BUILD/10-build_opensvc.sh`

Function signature: e4ac394e47db157e04fecc2f4b78c7140347675a65a982269bafad18d69af063

3.0.505 Function Overview

The `package` function primarily serves to copy the contents from the `startdir/usr` directory over to the `pkgdir`. It performs two actions: the first line creates the `pkgdir` directory (including any necessary parent directories) if it does not exist, while the second line utilizes the `cp -a` command, which not only copies files from one location to another, but also retains the links, ownership, timestamps, and permissions of the files being copied.

3.0.506 Technical description

- Name: `package`
- Description: This function creates the directory “pkgdir” if it doesn’t exist and proceeds to copy all the content from “startdir/usr/” directory into the “pkgdir”.
- Globals: [`pkgdir`: Destination directory for files, `startdir`: Source directory for files]

- Arguments: No direct arguments are required for this function.
- Outputs: The function outputs to the file system, creating a directory and duplicating files from one directory into another.
- Returns: No value, as it performs operations on the file system.
- Example usage:

```
startdir=/path/origin_directory
pkgdir=/path/destination_directory
package
```

3.0.507 Quality and Security Recommendations

1. A check should be implemented to ensure that `startdir` and `pkgdir` have been specified and they exist. In its current form, the function would throw an error if these variables are not set before running the function.
2. It is highly suggested to handle the case where either the `pkgdir` or `startdir/usr` directory does not exist. For instance, testing if the `startdir/usr` directory exists before attempting to copy from it could prevent a potential error.
3. Applying appropriate permissions to the `pkgdir` directory could enhance security. It's crucial to consider who should have access to view, write, and execute the files within the directory.
4. It would be prudent to consider adopting a logging strategy. By maintaining a record of activity, especially any errors, it is much easier to troubleshoot should something unexpected happen.

3.0.508 `run_function_list`

Contained in `node-manager/alpine-3/TCH/BUILD/run_osvc_build.sh`

Function signature: 9332f51df9c8b0d77c5f2b16e6f5e98293a660ace81153ea206e9ab1b86c0e9c

3.0.509 Function overview

The `run_function_list()` function is a bash function used to run a series of functions in sequence, providing logging for each function's execution and stopping the sequence if any function returns a non-zero exit code.

3.0.510 Technical description

- **Name**: `run_function_list`
- **Description**: This function iterates through an array of function names provided as arguments. Each function is run in sequence, and the execution and exit codes

of these functions are logged. If any function returns a non-zero exit code, the sequence is immediately stopped and the non-zero exit code is returned.
- **Globals**: None.
- **Arguments**: This function accepts an unlimited number of arguments. Each argument is expected to be a string corresponding to a function name to run.
- **Outputs**: This function outputs logs to `stdout` containing the name of each function as it is run and whether it was successful or failed.
- **Returns**: This function returns `0` if all functions run successfully. If any function fails, it returns the exit code of the failed function.
- **Example usage**: `bash ``run_function_list "function1" "function2" "function3"``

3.0.511 Quality and security recommendations

1. Functions provided to `run_function_list` should have proper error handling to ensure they exit with a non-zero status when an error occurs.
2. Ensure that all functions called are defined and have the correct permissions to run to avoid possibly sensitive errors being output and logged.
3. Avoid providing user-supplied or unsanitized input as function names to this function to prevent potential code injection attacks.
4. Be careful with the number of arguments. Massive numbers may lead to unexpected behaviour due to argument length limitations.
5. I would recommend adding measures to this function to handle cases where a function could potentially get stuck in an infinite loop.

3.0.512 `start_pre`

Contained in `node-manager/alpine-3/TCH/BUILD/10-build_opensvc.sh`

Function signature: 84093d24ed6e059f89dd6ad32c2ff11b16b0fd2540a9f3c88ab3932e8a93570b

3.0.513 Function overview

The `start_pre` function is used to prevent the current process from being targeted by Linux's out-of-memory (OOM) killer by setting its score to the minimum possible value first. Then, it checks whether the directory `/var/lib/opensvc` exists. If not, it creates the necessary directories.

3.0.514 Technical description

- **Name**: start_pre
- **Description**: This function lowers the OOM score of the current process and creates a directory if necessary.
- **Globals**: None

- **Arguments**: None
- **Outputs**: If successful, the function will reduce the process's likelihood of being killed during an out-of-memory condition and will create the directory `/var/lib/opensvc` if it isn't already present.
- **Returns**:
  - Writes `-1000` to `/proc/self/oom_score_adj` to lower the process's OOM score
  - Creates the directory `/var/lib/opensvc` if it does not exist
- **Example usage**: `start_pre`

3.0.515 Quality and security recommendations

1. In the current state, the function is assumed to have enough permissions to write to files and create directories wherever required. Consider adding error handling where it checks for the necessary permissions before attempting operations.
2. For the overall security, avoid running scripts with super-user rights when not necessary. Regularly monitor and audit all the activities running under root.
3. It's recommended to take measures in avoiding OOM conditions in general rather than adjusting OOM scores of individual processes, as OOM score is only one of many factors the kernel considers when deciding which process to kill. The best solution would be to ensure the system has sufficient memory for its workload.

3.0.516 `storage_deprovision_volume`

Contained in `node-manager/rocky-10/storage-management.sh`

Function signature: 668ca90d247ae9b85bd28558e102cecbf0dd119fbbf5fca3b61e894cf67082d7

3.0.517 Function overview

The `storage_deprovision_volume()` function written in BASH is designed for volume management in a storage cluster. It provides an interface to deprovision storage volumes on storage hosts. It takes two arguments: a unique identifier (`iqn`) and the volume name (`zvol_name`), then performs several checks, including confirmations that the host is a storage host and the local pool name exists. If all required parameters are valid, it deletes the iSCSI target specified by `iqn` and the volume specified by `zvol_name` from the ZPOOL.

3.0.518 Technical description

```
- __Name__: `storage_deprovision_volume`
- __Description__: De-provision (delete) iSCSI target and volume
 ↪  in storage host given `iqn` and `zvol_name`. Checks for
 ↪  necessary settings and fails with error message if required
 ↪  parameters are missing.
```

```
- __Globals__: None.
- __Arguments__:
  - `$1 (--iqn)`: iSCSI Qualified name, unique identifier for the
 ↪  iSCSI target.
  - `$2 (--zvol-name)`: The name of the volume to be
 ↪  de-provisioned.
- __Outputs__: Logs information about the deletion process,
 ↪  including success or failure messages.
- __Returns__:
  - `0` if the volume deprovisioning was successful.
  - `1` if there was a failure at any point during the
 ↪  deprovisioning process.
- __Eexample usage__: `storage_deprovision_volume --iqn
 ↪  iqn.2022-02.my.host:my.volume --zvol-name my_volume`
```

3.0.519 Quality and security recommendations

1. Input validation should be more robust. Currently, the function merely checks if the parameters are not empty; it doesn't validate if the supplied `iqn` and `zvol-name` parameters are of the correct format.
2. Consider creating secure logging methods. While logging is vital for debugging, it may potentially expose sensitive information in clear text.
3. Develop a mechanism to handle unexpected errors or exceptions during the function execution, with appropriate error messages and exit codes for easier troubleshooting.
4. The script should follow a consistent code style to increase readability and maintainability.
5. For high-security environments, consider integrating this command with a higher privilege level system instead of directly interacting with underlying system commands.

3.0.520 `storage_get_available_space`

Contained in `node-manager/rocky-10/storage-management.sh`

Function signature: 0ecd662d000f3348b1348219cc1c7541ead05b93b29de5edfc244de079f38d03

3.0.521 Function Overview

The function `storage_get_available_space()` is designed to obtain the available memory space in a given storage pool. The function begins by identifying the storage pool from a remote host, and if it fails to determine it, it will log an error and terminate the process. If it successfully identifies the pool, it will then attempt to retrieve the available memory space expressed in bytes. Again, if it encounters an issue at this step, it will log an error and terminate. Otherwise, it will display the available memory space and successfully end the process.

3.0.522 Technical Description

Name: `storage_get_available_space()`
Description: This function retrieves and displays the available storage space from a specified storage pool, in bytes.
Globals: `[ ZPOOL_NAME: Used to specify the pool of storage to check the available space ]`
Arguments: None
Outputs: Console output of the available storage space within the storage pool, in bytes. Logs an error message to the console if the ZPOOL_NAME cannot be determined or if there's an issue retrieving the available storage space.
Returns: `0` if the function successfully retrieves and displays the available memory space. `1` if the function fails to determine the ZPOOL_NAME, or if it's unable to get the available memory space.
Example Usage:

```
storage_get_available_space
```

3.0.523 Quality and Security Recommendations

1. Always sanitize and validate the ZPOOL_NAME. Ensure that it is not manipulated to carry out code injection attacks.
2. Introduce better error handling and logging. When an error is encountered, provide clearer messages that can help in diagnosing the issue.
3. Improve user permissions management. Make sure that only authorized users can retrieve the available memory space from storage pool.
4. Add functionality to handle any errors during transmission of data from the remote host. This will ensure reliable operation and robust error handling.
5. Implement more thorough testing for this function to identify and correct any possible weaknesses or bugs.

3.0.524 `storage_parse_capacity`

Contained in `node-manager/rocky-10/storage-management.sh`

Function signature: c67957f8ee4be8442088f8824ef3828b5a9b2d67c4b4f352df2b050bf6c8bbee

3.0.525 Function overview

This Bash function named `storage_parse_capacity` is designed to convert human-readable storage size specifications (e.g., 10K, 10M, 10G, 10T) into real storage size in bytes. The function accepts a string as input, which should represent a storage capacity with an optional suffix (K, M, G, T) corresponding to kilobytes, megabytes, gigabytes, or terabytes. If no suffix is provided, the function treats the input as representing bytes.

3.0.526 Technical description

- **Name:** storage_parse_capacity
- **Description:** This function converts a string indicating storage capacity with potential kilobytes, megabytes, gigabytes, or terabytes suffixes into an exact number of bytes.
- **Globals:** None
- **Arguments:** The function accepts two arguments -
  - $1: The input string representing a storage capacity with an optional suffix (K, M, G, T)
- **Outputs:** The function, upon success, outputs a string representing the storage capacity in bytes.
- **Returns:** The function returns '0' upon successful execution and '1' in case an error occurs, or the input value is not recognized.
- **Example usage:**

```
storage_parse_capacity "10G"
# Outputs: "10737418240"
```

3.0.527 Quality and security recommendations

1. The function should validate if the input value is a positive integer before proceeding. This can help prevent unexpected outcome or failure.
2. A verification system could be added to confirm whether the suffix K|M|G|T is used appropriately. This would prevent misuse where someone might input a wrong suffix by mistake.
3. Consider implementing error messages to notify the user of the specific error that has occurred. This would improve user-friendliness and facilitate troubleshooting.
4. This function does not differentiate between upper-case and lower-case suffixes for capacity. Be consistent in your codebase and user documentation about using either upper-case or lower-case.
5. Make sure potential security implications are always considered when dealing with user input, even when it seems harmless like in this case. Test this function rigorously with boundary values and unexpected values to ensure no security vulnerabilities are present.

3.0.528 `storage_provision_volume`

Contained in `node-manager/rocky-10/storage-management.sh`

Function signature: 5e3861e2975c7a31100612e49933846099de90b2882d09056b15392c4698cf6b

3.0.529 Function Overview

The function `storage_provision_volume` is designed to provision a storage volume on a storage host, creating a zvol and an iSCSI target. The function takes arguments

related to iSCSI Qualified Name (iqn), the capacity, and the zvol name. It uses various checks to ensure that all the required inputs are present and valid, and also that there is sufficient available storage space for the operation to occur.

3.0.530 Technical Description

- **Name**: `storage_provision_volume`
- **Description**: This function provisions a storage volume, conducting several validations, operations, and checks, such as parsing and checking arguments, validating host type and zpool name, calculating the available storage, creating zvol and iSCSI target.
- **Globals**: None.
- **Arguments**:
  - `$1`: iSCSI Qualified Name (iqn).
  - `$2`: The desired storage capacity for the zvol.
  - `$3`: The name of the zvol to be created.
- **Outputs**: Log messages signifying the different stages of the operation.
- **Returns**: This function will return `1` if any issues arise (such as missing parameters, inappropriate host type, insufficient space, etc.). If successful, the function will return `0`.
- **Example Usage**:

```
storage_provision_volume --iqn
 ↪  "iqn.2005-03.com.example:storage:diskarrays-sn-a8675309"
 ↪  --capacity "500GB" --zvol-name "zvol1"
```

3.0.531 Quality and Security Recommendations

1. Integrating error reporting or monitoring system, which could alert operators in the event of an error.
2. Implementing stricter parameter checks and verification, enhancing the validation process.
3. Including unit tests to confirm function behavior and to detect potential bugs.
4. Adding role-based checks to ensure the operation is executed by an authorized entity.
5. Implementing robust logging to provide transparency and traceability for all operations.

3.0.532 `zfs_get_defaults`

Contained in `node-manager/rocky-10/zpool-management.sh`

Function signature: 7075829ac859fa1a3d095289b6f87eeae5bfd9df08c9d1cfd66df7de132cf128

3.0.533 Function Overview

The `zfs_get_defaults` function in bash is used to initialize two sets of options POOL_OPTS and ZFS_PROPS for ZFS filesystem. The first set of options _POOL_OPTS is assigned configuration options for pool creation. Further, it considers how zpool can effectively create on disk, supporting extra -O props. The other _ZFS_PROPS is a set of default properties for the ZFS filesystem like logs compression, access control lists (ACL), and greater throughput, etc.

3.0.534 Technical Description

Define the zfs_get_defaults block for pandoc as follows:

- Name: `zfs_get_defaults`
- Description: Initializes two sets of properties for ZFS filesystem, _POOL_OPTS for pool options and _ZFS_PROPS for ZFS properties
- globals: None
- Arguments:
  - $1: pointer to an associative array _POOL_OPTS to store ZFS pool options
  - $2: pointer to an associative array _ZFS_PROPS to store ZFS filesystem properties
- Outputs: Initializes the _POOL_OPTS and _ZFS_PROPS
- Returns: None
- Example usage: `zfs_get_defaults POOL_OPTS ZFS_PROPS`

3.0.535 Quality and Security Recommendations

1. Always validate the input parameters to the function. Checking that the inputs are both defined would prevent potential script errors.
2. Commenting on each option can make it easier for others to understand the reasoning behind some options used.
3. Make use of bash's built-in facilities for ensuring that variables are set and not empty. This can prevent potential problems if an expected variable is not set.
4. Confidential information, like passwords or secret keys, should not be hard-coded into scripts but should be passed securely through environment variables or secure files.
5. Always maintain the function modularized and clean for scalability and easier debugging.
6. Make sure to escape any output that is included in generated HTML to avoid cross-site scripting (XSS) attacks. Not relevant in this function as it doesn't generate any HTML, but it's good to keep in mind for functions that do.

3.0.536 `zpool_create_on_free_disk`

Contained in `node-manager/rocky-10/zpool-management.sh`

Function signature: ff0bf00dc7c2ed8816d8a88a6b148a5993ba6e4c5d1ae70415c5960612576a80

3.0.537 Function overview

The `zpool_create_on_free_disk` function is a BASH script function that creates a zpool (Zettabyte File System pool) on the first free disk it identifies. It utilizes a hostname to return the network node's host name and subtracts any domain information from it (only retains the short version of the hostname).

3.0.538 Technical description

- **Name:** `zpool_create_on_free_disk`
- **Description:** This function creates a zpool on a free disk. It uses a "first come, first served" strategy by default, but this can be modified. It's designed to mount at "/srv/storage". There are flags that allow the function to force creation, do a dry run, and whether or not to apply defaults. The hostname is truncated to its short version for usage in the function.
- **Globals:**
  - `strategy`: It defines the strategy of the function, here set as "first".
  - `mpoint`: It specifies the default mount point for zpool.
  - `force`: It defines whether the function forcibly creates a zpool or not.
  - `dry_run`: It illustrates whether the function performs a dry run.
  - `apply_defaults`: It indicates whether the function applies the default settings.
  - `host_short`: It represents the short hostname of the system where the function is run.
- **Arguments:** No direct arguments are passed to this function.
- **Outputs:** Outputs are not explicitly defined in function snippet provided.
- **Returns:** Return value not explicitly defined in function snippet provided.
- **Example usage:** As the function snippet provided does not contain any direct arguments or a return statement, there is not enough information for an example usage.

3.0.539 Quality and security recommendations

1. Define return types for better error handling - this would greatly improve function use in larger scripts.
2. The function might need some argument validation if it is going to accept arguments in the future.
3. Include input sanitization particularly if the function is to be exposed to untrusted users or used in larger scripts where the data passed into it is not trusted.
4. Error checks after significant function calls (e.g., creating zpool) would be good for ensuring stability and data integrity.

5. It could be more secure to not store hostname in a global variable if it's not used often, and just call `hostname -s` directly in the code when necessary.

3.0.540 `zpool_name_generate`

Contained in `node-manager/rocky-10/zpool-management.sh`

Function signature: 5995908b1c71b7b0931db1a09cf94c2257d6d0ed783b7e45ca8926f62b975cf6

3.0.541 Function Overview

The function `zpool_name_generate` is a bash function that generates the name of a zpool, given the class of the zpool as an argument. This function also generates a timestamp and a random string, it then concatenates all these into a formatted string to create a unique zpool name. The function can generate names for five classes of zpools: `nvme`, `ssd`, `hdd`, `arc`, `mix`.

3.0.542 Technical Description

- **name:** `zpool_name_generate`
- **description:** Generates a zpool name based on the provided class, current timestamp, and a random string. The name is generated in the format: `z${cluster}-p${class}-u${secs}${rand}`
- **globals:** [`CLUSTER_NAME`: The name of the cluster]
- **arguments:** [`$1`: The class of the zpool. The options are: `nvme`, `ssd`, `hdd`, `arc`, `mix`]
- **outputs:** Prints the generated zpool name or an error message in case of an invalid class or incorrect usage.
- **returns:** Returns 2 if the usage is incorrect or the class is invalid.
- **example usage:** `zpool_name_generate hdd`

3.0.543 Quality and Security Recommendations

1. Error messages should be made more descriptive
2. Consider throwing exceptions or exiting when the class is empty or invalid instead of returning error codes.
3. Potential validations for input arguments.
4. Robust error handling and logging is recommended.
5. Name generation module can be improved for uniqueness and predictability.
6. Consider using secure random generators to prevent potential security risks.
7. A code review for potential command injection vulnerabilities is recommended.
8. Attention must be paid to the permissions of scripts and the permissions of the users or processes that will use these functions to prevent unauthorized access.

3.0.544 zpool_slug

Contained in node-manager/rocky-10/zpool-management.sh

Function signature: 18bbbffc19d9426de745e5b45fe837f9d50ec9443122924bb19421975f877cbc

3.0.545 Function overview

The zpool_slug function in Bash takes a string input and modifies it to meet certain criteria. It first converts all the characters in the string to lower case. Then, it removes any characters that are not lower-case letters, numbers or hyphens. It also replaces any occurrence of two or more consecutive hyphens with a single one. Lastly, it trims off any hyphen(s) at either the start or the end of the string. It also provides an optional second argument which limits the length of the output string. The function uses printf to return the final result, effectively modifying the input string to a format similar to URL slugs.

3.0.546 Technical description

- **Name**: zpool_slug
- **Description**: This function converts a string into a slug-like format, making it lower case, replacing non-alphanumeric, non-hyphen characters with hyphens, trimming leading or trailing hyphens, and limiting the length of the output if needed.
- **Globals**: None
- **Arguments**:
  - $1: The input string to be converted to a slug format
  - $2: An optional argument that specifies the maximum allowable length of the output string, defaulting to 12 characters if not provided
- **Outputs**: The modified slug-like string
- **Returns**: None
- **Example usage**: zpool_slug "Hello World" 10 would return hello-world

3.0.547 Quality and security recommendations

1. Always validate input: In general, any function that accepts input should validate that input before processing it. Although this function is quite simple, accepting a wider range of characters for conversion could inadvertently allow injection of malicious commands.
2. Handle errors: This function does not currently have any error handling, such as checking if the first argument is given, or confirming that it is a string. Consider adding checks and error messages to inform the user if the function is used incorrectly.

3. Consider whether `printf` is the most effective choice: While using `printf` here works well to limit the final string length, it does not provide any feedback if the string is truncated. Depending on context, it might alternatively be desirable to throw an error when truncation occurs.

4 Deploying the disaster recovery node

Installing and configuring the DR node, synchronising data, and testing recovery procedures. ## Deployment process

Stub: Steps for installing and integrating the DR node into the cluster. ## Purpose of the DR node

Stub: Explain the role of the DR node in ensuring service continuity. ## Failover and recovery testing

Stub: Procedures for verifying DR readiness and simulating failover scenarios. ## Preparation

Stub: Hardware, storage, and network prerequisites for the DR node. ## Synchronisation

Stub: Methods for keeping DR node data in sync with the primary environment. # Design Decisions

4.1 Choice of boot firmware: UEFI vs legacy BIOS for diskless servers

Decision **Legacy BIOS** selected as the boot firmware mode for diskless servers.

4.1.1 Rationale:

- Using UEFI with iSCSI-backed ZFS volumes in a RAID1 configuration was found to add **significant complexity** to the boot process, especially around firmware-level RAID support and driver requirements.
- Legacy BIOS booting is **simpler and more predictable** in a diskless environment, particularly when boot devices are provided as iSCSI LUNs managed at the client level with MD RAID.
- Secure Boot provides little additional value in this scenario since the root disk is remote (iSCSI) and already subject to provisioning controls in HPS, making the complexity of UEFI+Secure Boot unjustified.
- By avoiding UEFI, the system benefits from a **cleaner, more robust boot path** with fewer moving parts, reducing risk during provisioning and recovery.

4.1.2 Trade-offs and compromises:

- Some modern hardware platforms are increasingly UEFI-centric, and relying on BIOS boot mode may limit compatibility with certain devices in the future.
- Secure Boot cannot be leveraged for kernel validation in this model, which may be a requirement in highly regulated environments.
- Legacy BIOS lacks some advanced features of UEFI such as larger boot volume support and graphical configuration menus, though these are not critical for diskless servers.

4.1.3 Alternatives considered:

- **UEFI with Secure Boot** – Provides firmware-level kernel verification but introduces bootloader and driver complexity with limited real-world benefit for iSCSI-booted, diskless nodes.
- **UEFI without Secure Boot** – Simplifies compared to full Secure Boot but still adds no significant functional gain for this scenario, while retaining complexity.
- **Mixed mode** – Maintaining both BIOS and UEFI boot paths was considered but rejected as it increases maintenance burden without improving robustness.

4.1.4 Summary:

For diskless servers provisioned by HPS, **Legacy BIOS booting was chosen** as it avoids unnecessary UEFI complexity and delivers a simpler, more reliable boot process. Secure Boot does not meaningfully strengthen security in this architecture, where the root filesystem is centrally provisioned over iSCSI.

4.2 Choice of File System for iSCSI Export: ZFS vs. Btrfs

Decision ZFS selected as the primary file system for our iSCSI export volumes.

4.2.1 Rationale:

- **ZFS supports native block devices (zvols),** allowing direct and efficient exports of block storage over iSCSI. This feature aligns well with our requirement for reliable, performant SAN/NAS workflows.
- ZFS provides **native, atomic snapshots and cloning** of zvols, enhancing backup and provisioning capabilities crucial for production environments.
- Its robust data integrity features (checksumming, self-healing), advanced caching, and mature RAID implementations contribute to enterprise-grade reliability and performance.

4.2.2 Trade-offs and Compromises:

- There is a **license incompatibility** between ZFS and the Red Hat ecosystem, meaning ZFS packages are not officially supported or included in RHEL distributions.
- As a result, ZFS must currently be **built from source or installed from third-party repositories**, adding complexity to deployment and ongoing maintenance.
- This introduces a support risk and necessitates additional operational effort compared to fully integrated Linux-native options.

4.2.3 Alternatives Considered:

- **Btrfs** was a close second due to its strong native support for snapshots, flexibility in resizing and tuning volumes on the fly, and better Linux kernel integration.
- However, **Btrfs lacks native block device export (no zvol equivalent),** forcing us to use file-backed iSCSI LUNs that generally perform less optimally and do not offer atomic snapshot capabilities at the block layer.
- Other Linux file systems (XFS, Ext4, Stratis) were evaluated but do not meet the combination of flexibility, native snapshot, and block device export requirements.

4.2.4 Summary:

While recognizing the complexity introduced by licensing and packaging constraints, **ZFS was chosen because it directly meets the core technical needs of efficient iSCSI block exports, robust snapshotting, and enterprise reliability.** Btrfs remains a strong alternative for scenarios prioritizing Linux-native integration and flexibility but is currently less ideal for our iSCSI workload due to the lack of native block device support.

This decision may be revisited as OpenZFS support for RHEL 10 matures or new technologies emerge.

4.3 Choice of Base OS Deployment Method: Pre-Built Image vs. Fresh Install

Decision **Fresh install** selected as the primary method for deploying the base operating system.

4.3.1 Rationale:

- **Fresh installs ensure hardware compatibility** by using the installer's latest kernel and driver set, reducing the risk of missing support for new or varied hardware configurations.

- Installing at provisioning time allows **security updates and package fixes** from the local mirror or upstream sources to be applied immediately, avoiding staleness issues inherent in pre-built images.

- The process starts from a **clean, reproducible configuration** defined by Kickstart or Preseed, eliminating unintended artifacts, logs, or credentials that may remain in captured images.

4.3.2 Trade-offs and Compromises:

- Fresh installs are **slower** than restoring a pre-built image, especially on large systems or when provisioning many hosts concurrently.

- The process **relies on functioning installer infrastructure** (DHCP/TFTP/HTTP and repository availability) at the time of deployment.

- There is **less bit-for-bit uniformity** compared to image-based deployment, as package versions may vary if mirrors are updated between installs.

4.3.3 Alternatives Considered:

- **Pre-built images** offer faster deployment and consistent results but can quickly become outdated, require additional effort to rebuild for each hardware or OS variant, and carry a higher risk of incompatibility if built on different hardware.

- Pre-built images may also include **unwanted residual configuration or data** unless carefully cleaned before capture, increasing operational risk.

- A **hybrid approach**—maintaining a fresh-install workflow while offering image duplication for identical hardware—was identified as a potential enhancement for high-speed redeployment in specific scenarios.

4.3.4 Summary:

While acknowledging the speed advantages of image-based deployment, **fresh installs were chosen for their hardware adaptability, up-to-date software, and clean configuration state**. The reduced risk of compatibility issues and the ability to apply updates during provisioning outweigh the longer install time in our current environment.

We will explore the feasibility of adding a controlled image duplication process to complement the standard fresh-install workflow for cases where rapid redeployment on identical hardware is beneficial.

4.4 Choice of Operating System: Alpine Linux vs Rocky Linux for SCH

Decision **Alpine Linux** selected as the operating system for SCH and TCH HPS node types.

4.4.1 Rationale:

- Rocky Linux kickstart was found to add significant opacity to the installation process, with limited visibility into %pre/%post script execution, silent errors, and difficult debugging cycles requiring full reboots (~5+ minutes each).
- Alpine Linux provides a simpler and more transparent boot and installation model, with direct netboot to a shell environment where state can be inspected at any point.
- TCH nodes already run Alpine successfully, and using the same OS for SCH enables consistent function loading, init sequences, and IPS communication across all node types.
- Alpine's fast boot time (seconds vs minutes) provides immediate feedback during development, dramatically reducing iteration time.
- The appliance philosophy of HPS aligns with Alpine's embedded/minimal design, where users should not need to care about the underlying OS.

4.4.2 Trade-offs and compromises:

- Alpine uses musl libc rather than glibc, which may introduce minor compatibility differences for some software, though this does not affect HPS use cases.
- Enterprise documentation and community resources for storage workloads on Alpine are smaller than for RHEL-based distributions.
- RHEL ecosystem familiarity and enterprise certifications are lost, though these are not critical for HPS's appliance model.

4.4.3 Alternatives considered:

These are intended to be revisited at a later date, as HPS should allow user preference of O/S

- **Rocky Linux with kickstart** – Provides RHEL ecosystem familiarity but introduces installer opacity, fragile script environments across %pre/%post sections, and slow debug cycles. Rejected due to excessive complexity.
- **Debian/Ubuntu** – Not considered as there is no existing HPS implementation and footprint is larger than Alpine.
- **Mixed OS environment** – Running Rocky on SCH and Alpine on TCH was rejected as it doubles maintenance burden and prevents sharing of node functions.

4.4.4 Summary:

For all HPS node types, Alpine Linux was chosen as it provides transparency, fast iteration, and consistency across TCH and SCH nodes. Rocky kickstart's opacity made it unsuitable for HPS's tight integration with IPS, where debugging visibility and predictable behaviour are essential.

5 HPS Network Topology Design

5.1 Objectives and Intent

5.1.1 Core Objectives

Flexible Deployment Stages Support deployment from simple single-NIC setups (testing/POC) through to production multi-NIC configurations without changing logical network definitions.

Infrastructure Isolation Separate HPS infrastructure networks (management, storage, VXLAN transport) from customer workload networks completely. Customer changes never impact HPS core systems.

Multi-Host Customer Networks Enable VMs and containers across different physical hosts to communicate on private customer networks without requiring switch VLAN configuration.

Bootstrap-Friendly Support diskless PXE boot while maintaining VLAN-based security model through rapid transition to tagged networks.

Simplicity and Robustness Minimize configuration complexity. Use standard Linux kernel features. Avoid proprietary protocols or complex control planes.

Scalability Start with minimal hardware for testing, expand to production-grade redundancy and performance without reconfiguration.

5.1.2 Design Intent

The HPS network design uses **VLAN abstraction at the infrastructure level** combined with **VXLAN overlays for customer networks**. This creates a clear separation:

HPS Infrastructure Networks Predefined VLANs (10, 20, 31, 32+) for management, VXLAN transport, and storage, mapped to physical interfaces based on deployment profile.

Customer Networks VXLAN overlays (VNI 1000+) providing isolated layer-2 domains spanning all host types (KVM, Docker, physical).

The bootstrap problem (diskless PXE requiring untagged network) is solved through a rapid transition approach:

1. Initial PXE boot on untagged network (~3-7 seconds exposure)
2. iPXE chainload configures VLAN 10 for management
3. All subsequent provisioning and runtime operations on VLAN 10

This ensures:

- Infrastructure stability (core VLANs remain unchanged)
- Customer flexibility (add/modify customer networks independently)
- Progressive enhancement (add hardware without reconfiguration)
- Minimal untagged network exposure (seconds, not minutes)

5.2 Network Architecture Overview

5.2.1 Infrastructure Networks (HPS Core)

```
┌──────────────────────────────────────────────────────────────┐
│                  HPS Infrastructure                          │
│                                                              │
│  VLAN 10: Management Network (192.168.10.0/24)               │
│    - SSH, monitoring, provisioning                           │
│    - Cluster coordination                                    │
│    - 1Gbps sufficient for Profile 2+                         │
│                                                              │
│  VLAN 20: VXLAN Transport Network (10.20.0.0/24)             │
│    - VXLAN multicast traffic                                 │
│    - Customer network encapsulation                          │
│    - Isolated from management                                │
│                                                              │
│  VLAN 31: Storage Network 1 (10.31.0.0/24)                   │
│    - iSCSI Target 1                                          │
│    - MTU 9000 (jumbo frames)                                 │
│    - 10Gbps recommended                                      │
│                                                              │
│  VLAN 32: Storage Network 2 (10.32.0.0/24)                   │
│    - iSCSI Target 2                                          │
│    - MTU 9000 (jumbo frames)                                 │
│    - 10Gbps recommended                                      │
│                                                              │
│  VLAN 33+: Additional Storage (10.33.0.0/24+)                │
│    - Scale storage capacity as needed                        │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```

5.2.2 VLAN Numbering Scheme

Reserved VLAN ranges to avoid:

- VLAN 0: Reserved (priority tagging)

- VLAN 1: Default VLAN (avoid for security)
- VLAN 1002-1005: Reserved (Cisco legacy protocols)

HPS VLAN allocation:

VLAN 10 Management (safe, commonly used)

VLAN 20 VXLAN Transport (safe)

VLAN 31-99 Storage networks (safe range, room for expansion) - 31: Storage 1 - 32: Storage 2 - 33-99: Additional storage hosts as needed

This numbering avoids all known reserved ranges and provides clear logical grouping.

5.2.3 Customer Networks (VXLAN Overlays)

```
┌──────────────────────────────────────────────────────────────┐
│                Customer Overlay Networks                     │
│       (Transported over VLAN 20 VXLAN Network)               │
│                                                              │
│  VNI 1000: Customer A Private (10.200.0.0/16)                │
│    - Multicast Group: 239.1.1.100                            │
│    - Spans: KVM hosts, Docker hosts                          │
│    - MTU 1450                                                │
│                                                              │
│  VNI 1001: Customer B Private (10.201.0.0/16)                │
│    - Multicast Group: 239.1.1.101                            │
│    - Isolated from Customer A                                │
│                                                              │
│  VNI 1002: Shared External Gateway (10.250.0.0/16)           │
│    - Multicast Group: 239.1.1.102                            │
│    - Firewall VMs provide internet access                    │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```

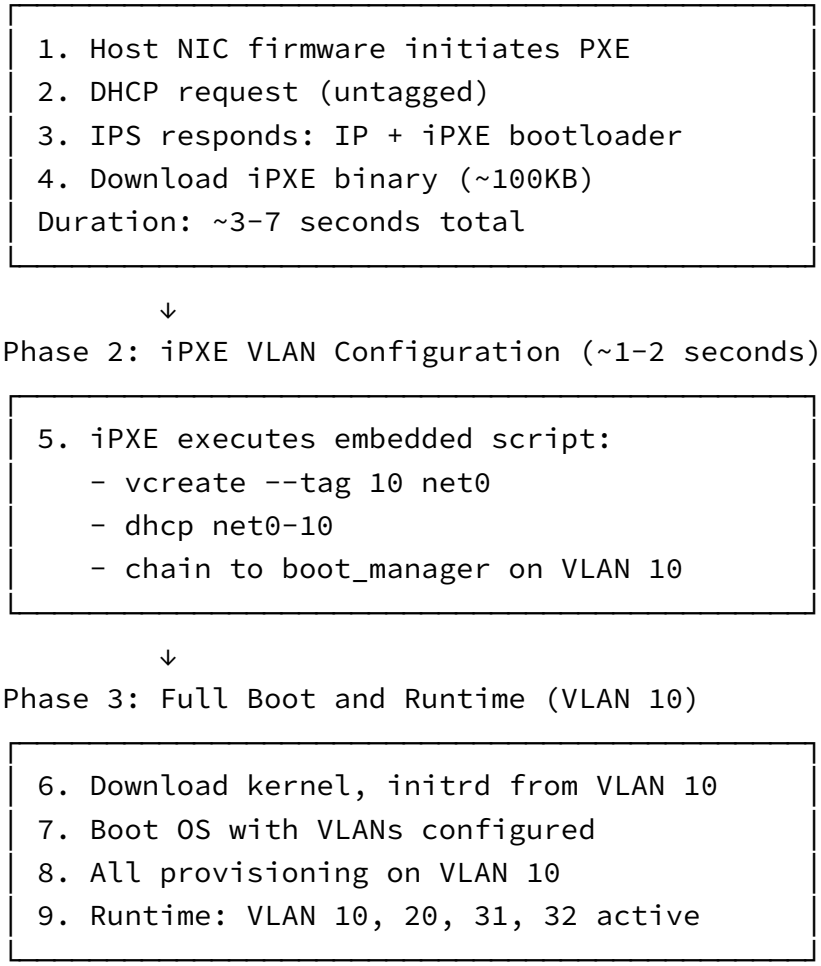
5.3 Bootstrap Process

5.3.1 The Challenge

Diskless hosts require network boot (PXE), but PXE firmware only supports untagged Ethernet. HPS runtime uses VLANs for security and isolation. This creates a bootstrap requirement.

5.3.2 HPS Solution: Rapid VLAN Transition

```
Phase 1: Initial PXE (untagged, ~3-7 seconds)
```



Untagged exposure: ~3-7 seconds

All runtime operations: VLAN-tagged

5.3.3 Bootstrap Security Modes

Exploratory Mode (Default - Profile 1) Untagged bootstrap enabled. Suitable for: Lab, testing, home, POC. Minimal security risk (seconds of exposure).

Production Mode (Profile 2+) Untagged bootstrap on dedicated 1Gb management NIC. Management traffic isolated from data networks. Suitable for: Production deployments.

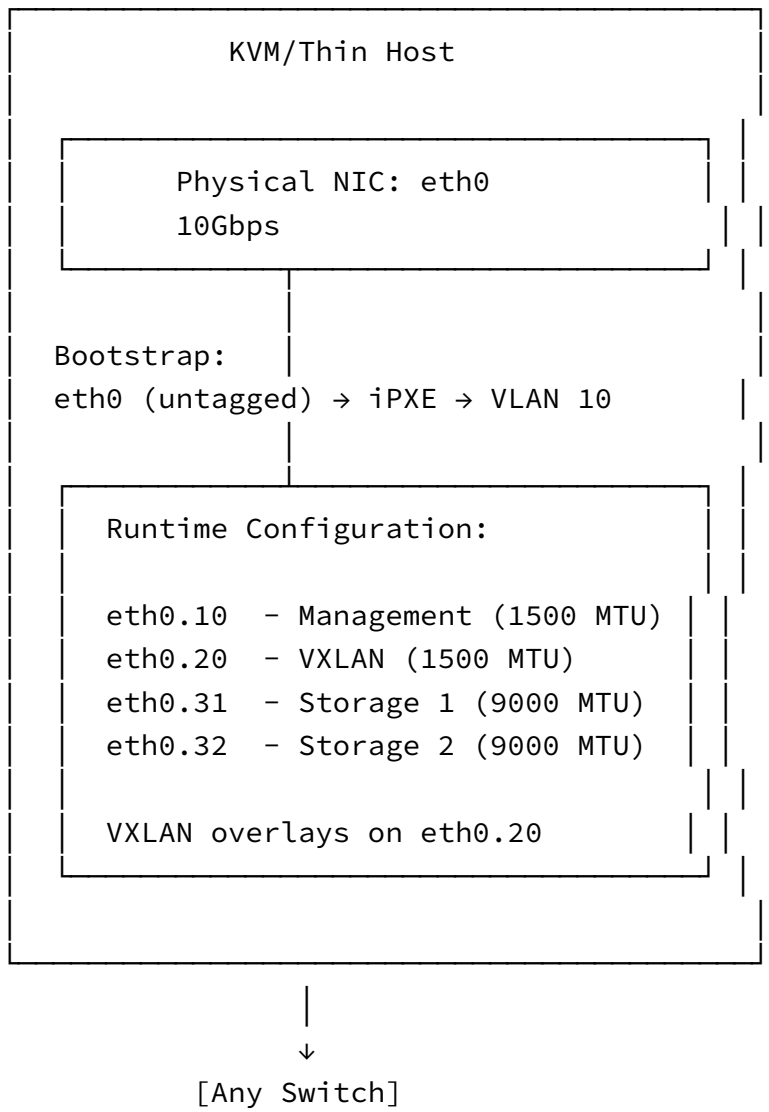
High Security Mode (Optional) No untagged bootstrap allowed. Requires: Pre-configured switch with VLAN 10 as native VLAN or out-of-band management (IPMI) for initial configuration. Suitable for: Highly regulated environments.

5.4 Deployment Stages

5.4.1 Profile 1: Single NIC (Testing/POC)

Hardware: 1 x 10G NIC per host, any switch (managed or unmanaged)

Purpose: Lab environments, home use, initial testing, proof of concept



Characteristics:

- All VLANs share single 10G interface
- Bootstrap: untagged (~3-7 seconds) → VLAN 10 transition
- Storage MTU 9000 (jumbo frames) on eth0.31, eth0.32
- Management/VXLAN MTU 1500 on eth0.10, eth0.20
- VXLAN multicast on eth0.20 (dedicated VLAN)
- Switch can be unmanaged for testing (inefficient but works)
- **Not performant - all traffic shares 10G - testing only**

Switch Requirements:

- None for unmanaged switch

- If managed: Trunk port with VLANs 10, 20, 31, 32 + allow untagged (bootstrap)

Bootstrap Process:

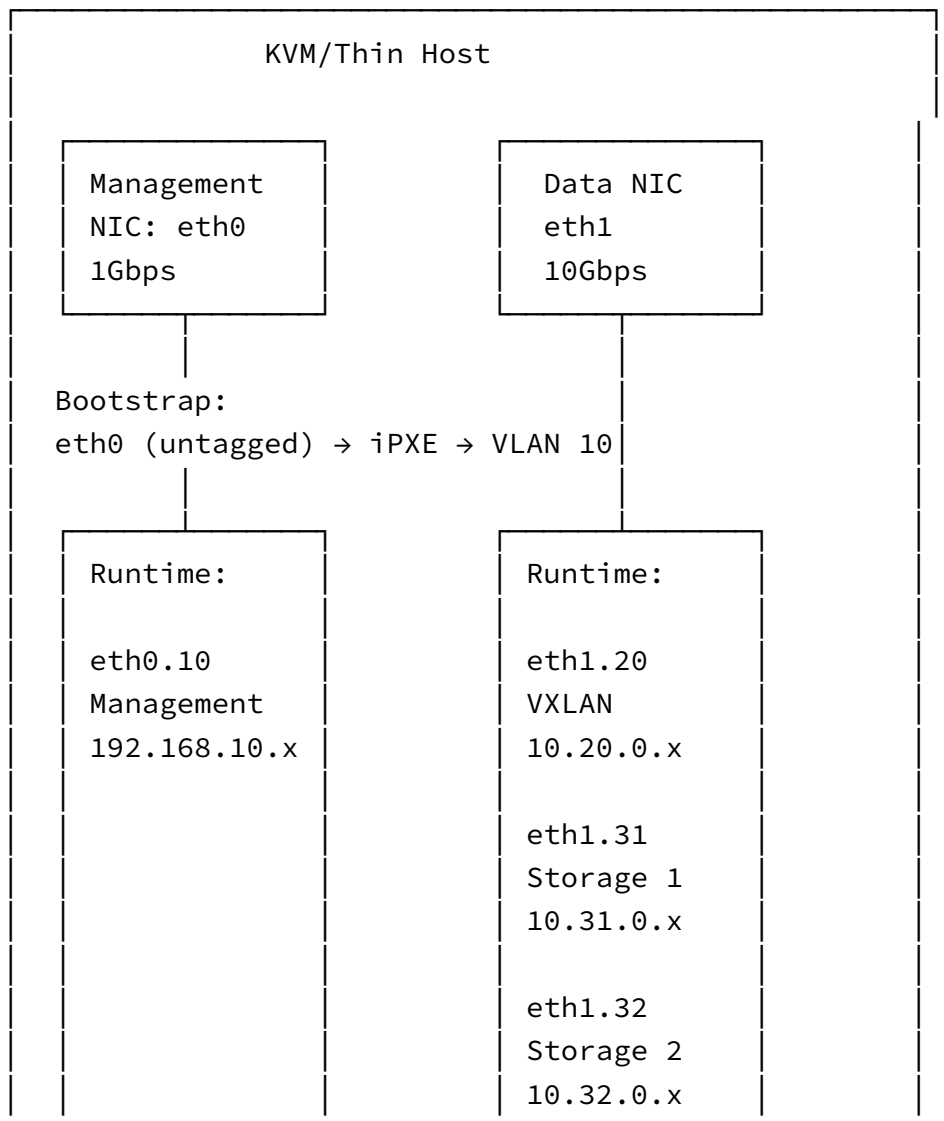
1. PXE boot (untagged) → iPXE
2. iPXE creates VLAN 10, gets IP
3. Downloads OS from VLAN 10
4. OS configures all VLANs (10, 20, 31, 32)

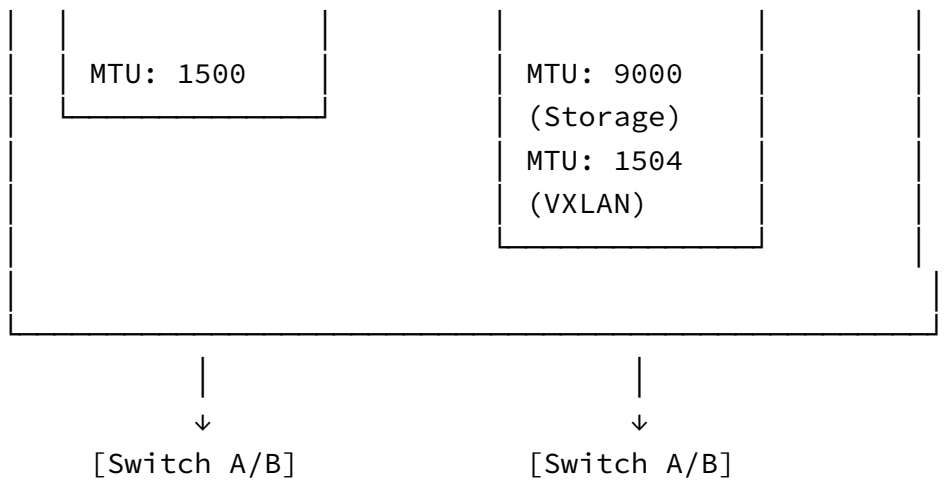
5.4.2 Profile 2: Dual NIC (Production Entry)

Hardware:

- 1 x 1G NIC (management)
- 1 x 10G NIC (VXLAN + storage)
- 1-2 managed switches

Purpose: Production deployments with network isolation





Physical Interface Assignment:

eth0 (1Gb) Management only (VLAN 10). SSH, monitoring, provisioning. Low bandwidth requirements. Dedicated interface for administrative access.

eth1 (10Gb) Data networks (VLAN 20, 31, 32). VXLAN transport (VLAN 20). Storage networks (VLAN 31, 32) with jumbo frames. High bandwidth for data operations.

Characteristics:

- Management isolated on dedicated 1Gb NIC
- VXLAN and storage share 10Gb data NIC
- Bootstrap on eth0 (untagged → VLAN 10 transition)
- Storage networks: MTU 9000 (jumbo frames for iSCSI)
- VXLAN network: MTU 1504 (accommodates VXLAN overhead)
- Management network: MTU 1500
- Clear separation: management vs data

Switch Requirements:

- Trunk ports supporting VLANs 10, 20, 31, 32
- IGMP snooping on VLAN 20 (recommended for VXLAN efficiency)
- Jumbo frame support on data ports (MTU 9000+)
- Allow untagged on management port (bootstrap)

Bootstrap Process:

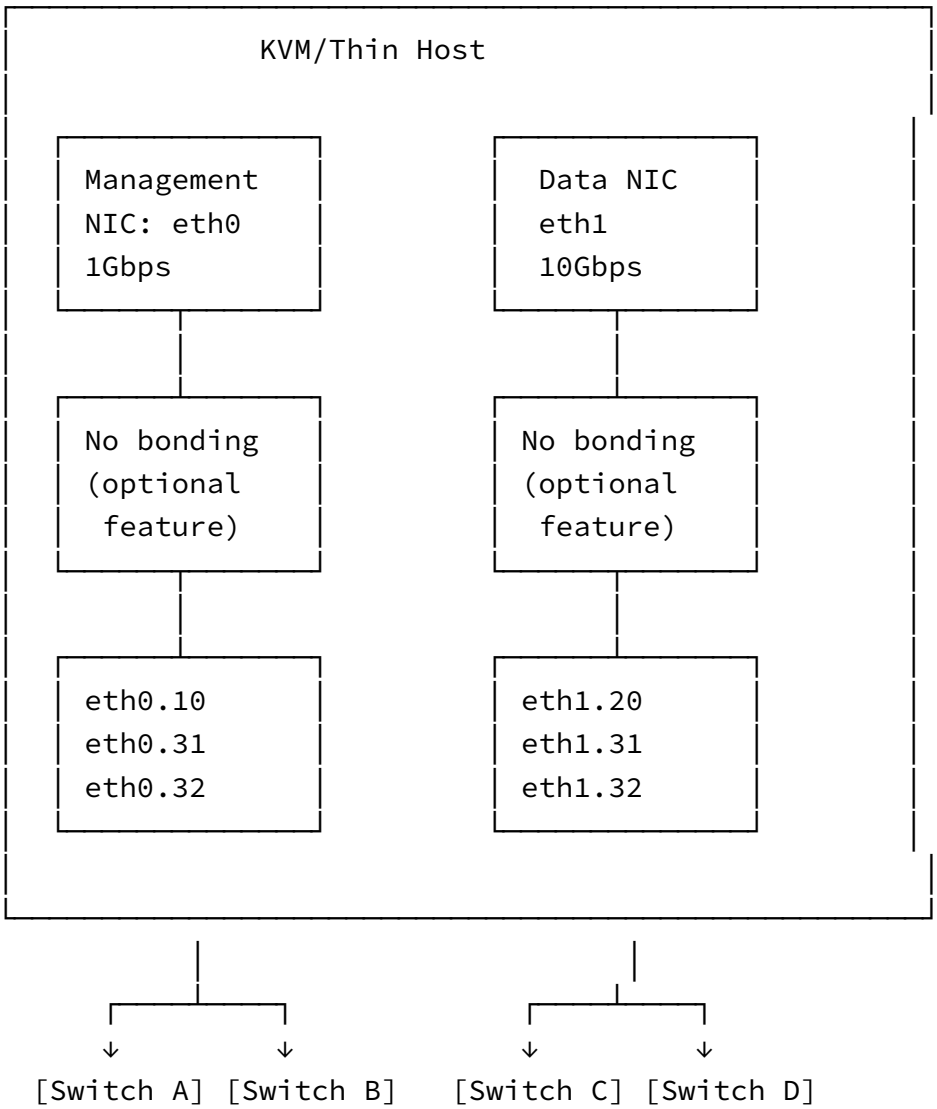
1. PXE boot from eth0 (untagged) → iPXE
2. iPXE creates eth0.10 (VLAN 10), gets IP
3. Downloads OS from VLAN 10
4. OS configures:
    - eth0.10 (management)
    - eth1.20 (VXLAN transport)
    - eth1.31, eth1.32 (storage)

5.4.3 Profile 3: Dual NIC, Dual Switch (High Availability)

Hardware:

- 1 x 1G NIC (management)
- 1 x 10G NIC (VXLAN + storage)
- 2 independent switches

Purpose: Production with switch-level redundancy



Optional Bonding Configuration:

Mode active-backup (no LACP required)
bond0 eth0 connects to Switch A and B (management failover)
bond1 eth1 connects to Switch C and D (data failover)
Failover Automatic upon link or switch failure

Characteristics:

- Switch-level redundancy (no single point of failure)
- Automatic failover if bonding used (typically <1 second)

- One active path per bond when bonding used (1G mgmt, 10G data)
- No switch coordination required (independent switches)
- Bonding optional - can run without for simpler configuration

Switch Requirements:

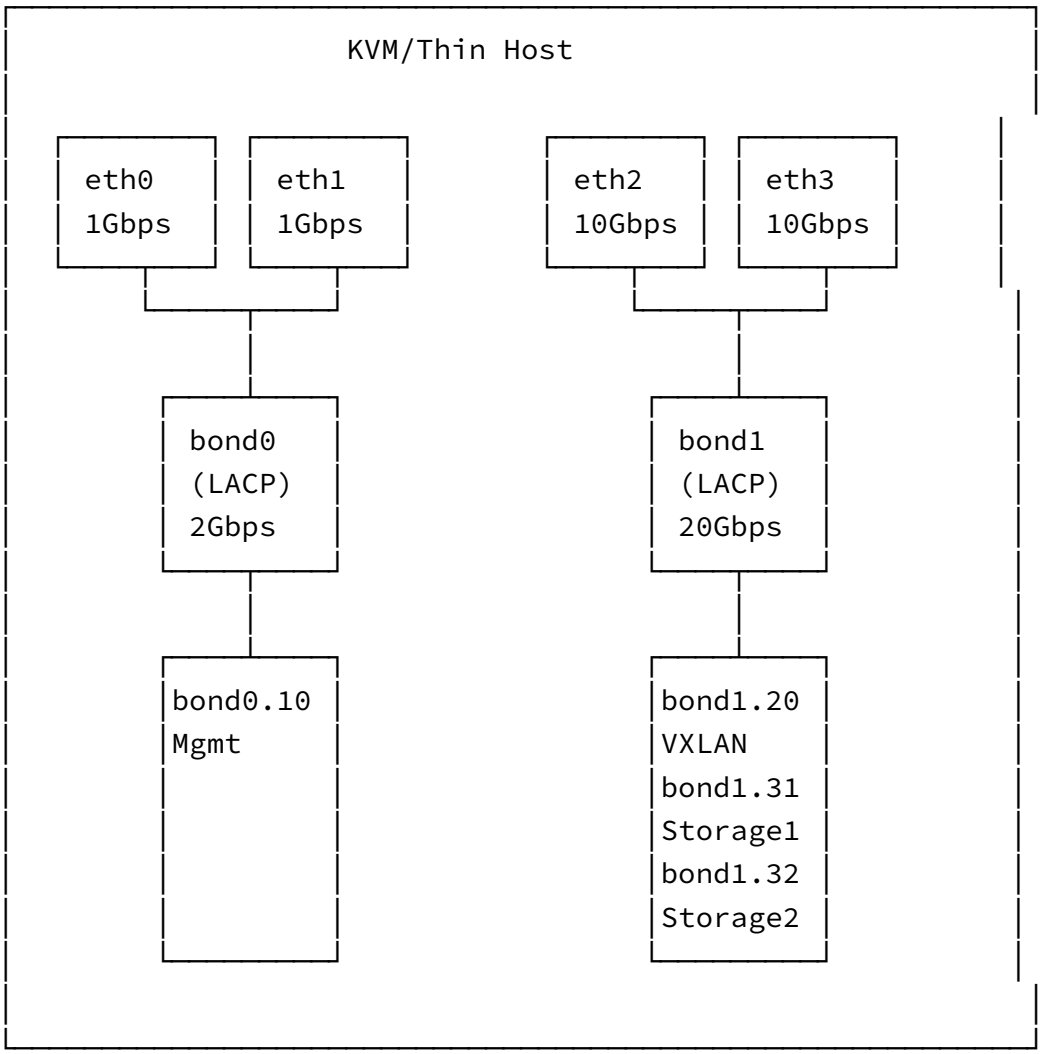
- Same as Profile 2
- Switches can be independent (no stacking/MC-LAG needed)
- If bonding used: same trunk configuration on both switches

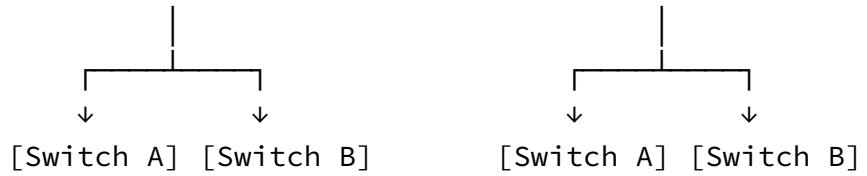
5.4.4 Profile 4: Quad NIC with LACP Bonding (Maximum Performance)

Hardware:

- 2 x 1G NICs (management bonding)
- 2 x 10G NICs (data bonding)
- 2 switches (can be independent)

Purpose: Production with bandwidth aggregation and redundancy





Note: Can use same switches or separate switch pairs for each bond

Bonding Configuration:

Mode 802.3ad (LACP)

bond0 eth0 + eth1 = 2Gbps aggregate (management). eth0 connects to Switch A, eth1 connects to Switch B.

bond1 eth2 + eth3 = 20Gbps aggregate (VXLAN + storage). eth2 connects to Switch A (or Switch C), eth3 connects to Switch B (or Switch D).

Hash policy layer3+4 (IP + port based distribution)

Characteristics:

- Maximum bandwidth: 22Gbps total (2G management + 20G data)
- Link-level redundancy within each bond
- Switch-level redundancy (each bond spans two switches)
- Load balancing across links (per-flow)
- Works with independent switches using standard LACP

Switch Requirements:

- LACP/802.3ad support (standard feature on managed switches)
- Each switch independently configured with LACP
- No switch stacking or MC-LAG required
- Traffic distributed per-flow between switches

Important: Each bond operates independently with standard LACP on each switch. This is the base Profile 4 configuration and works with any LACP-capable managed switches.

5.4.5 Optional Enhancement: MC-LAG for Profile 2+

MC-LAG (Multi-Chassis Link Aggregation) is an **optional enhancement** available for Profile 2, 3, or 4 when using advanced switches that support this feature.

MC-LAG allows two independent switches to coordinate and appear as a single logical switch for LACP purposes. This provides:

Benefits over standard LACP:

- Simpler host configuration (single bond instead of multiple independent bonds)
- Faster failover (sub-second vs 1-2 seconds)
- Active-active links (both links in bond actively used)

- No reconfiguration needed on switch failure

Requirements:

- Enterprise switches with MC-LAG capability
- Peer link between switches (high bandwidth)
- Vendor-specific configuration

Tradeoffs:

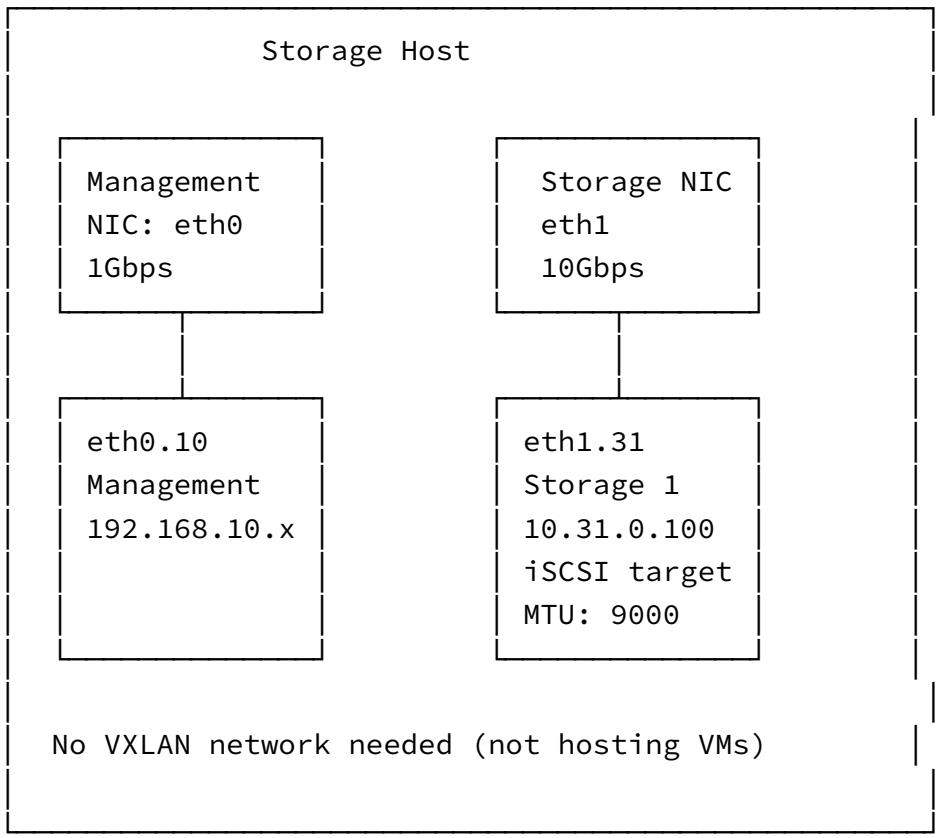
- More complex switch configuration
- Higher switch cost (enterprise feature)
- Vendor-specific setup

Recommendation: Start without MC-LAG using standard LACP (Profile 4 base configuration). This works with any LACP-capable managed switch and provides excellent redundancy and performance. Add MC-LAG later if you upgrade to enterprise switches with this capability and need sub-second failover for critical workloads.

5.5 Storage Host Considerations

5.5.1 Storage Host Network Configuration

Storage hosts (providing iSCSI targets) have similar but simplified networking:



VLANs used:

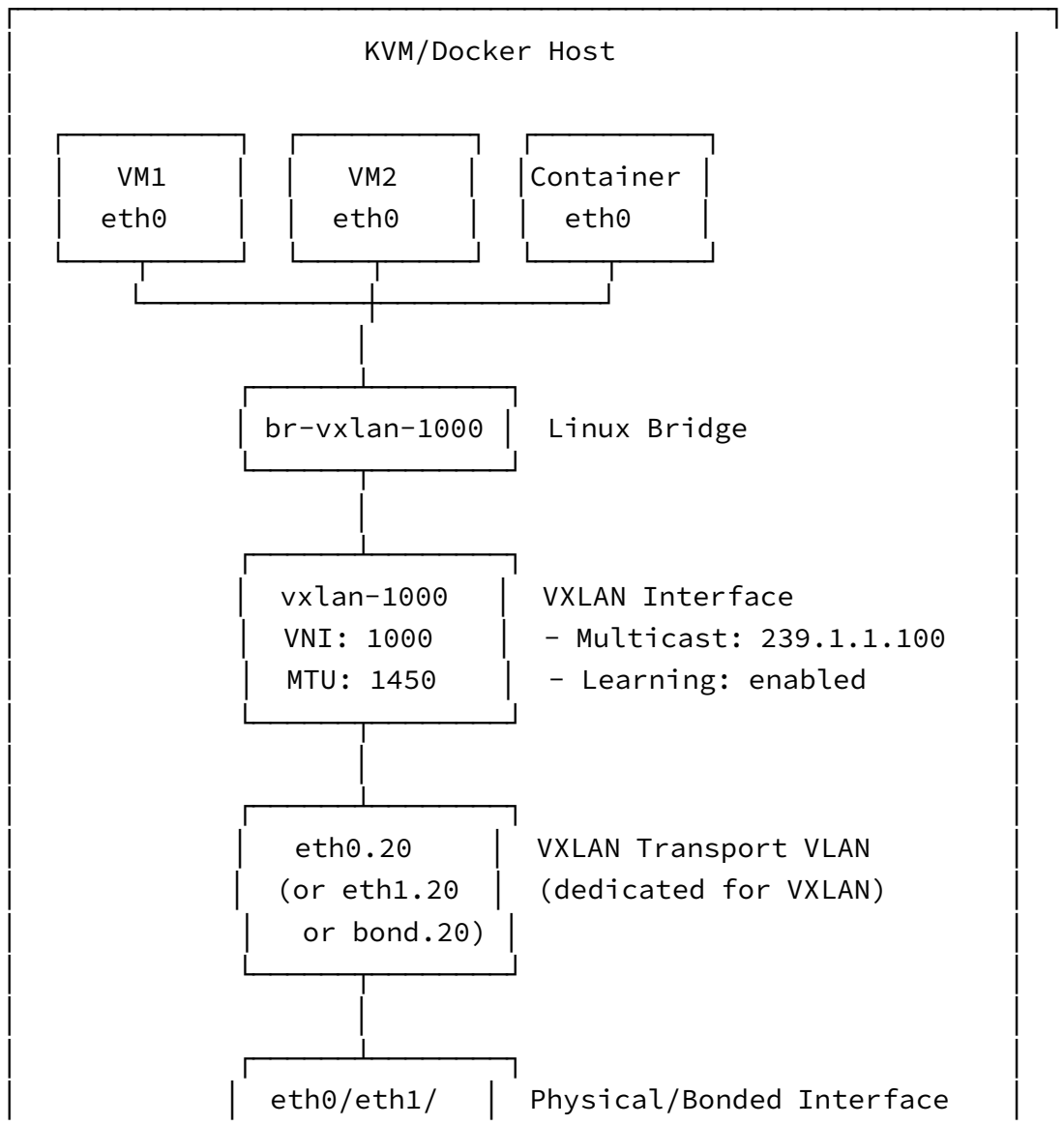
- VLAN 10: Management (SSH, monitoring)
- VLAN 31: Storage network for this host's iSCSI target
- No VLAN 20 (VXLAN) - storage hosts don't participate in customer networks

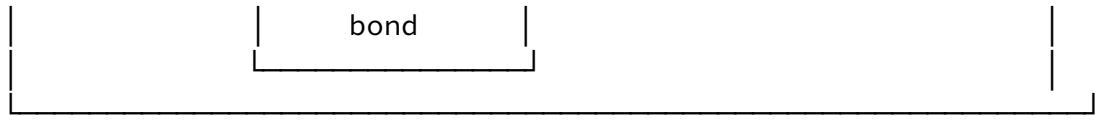
Each storage host uses one storage VLAN:

- Storage Host 1: VLAN 31 (10.31.0.100)
- Storage Host 2: VLAN 32 (10.32.0.100)
- Storage Host 3: VLAN 33 (10.33.0.100)
- etc.

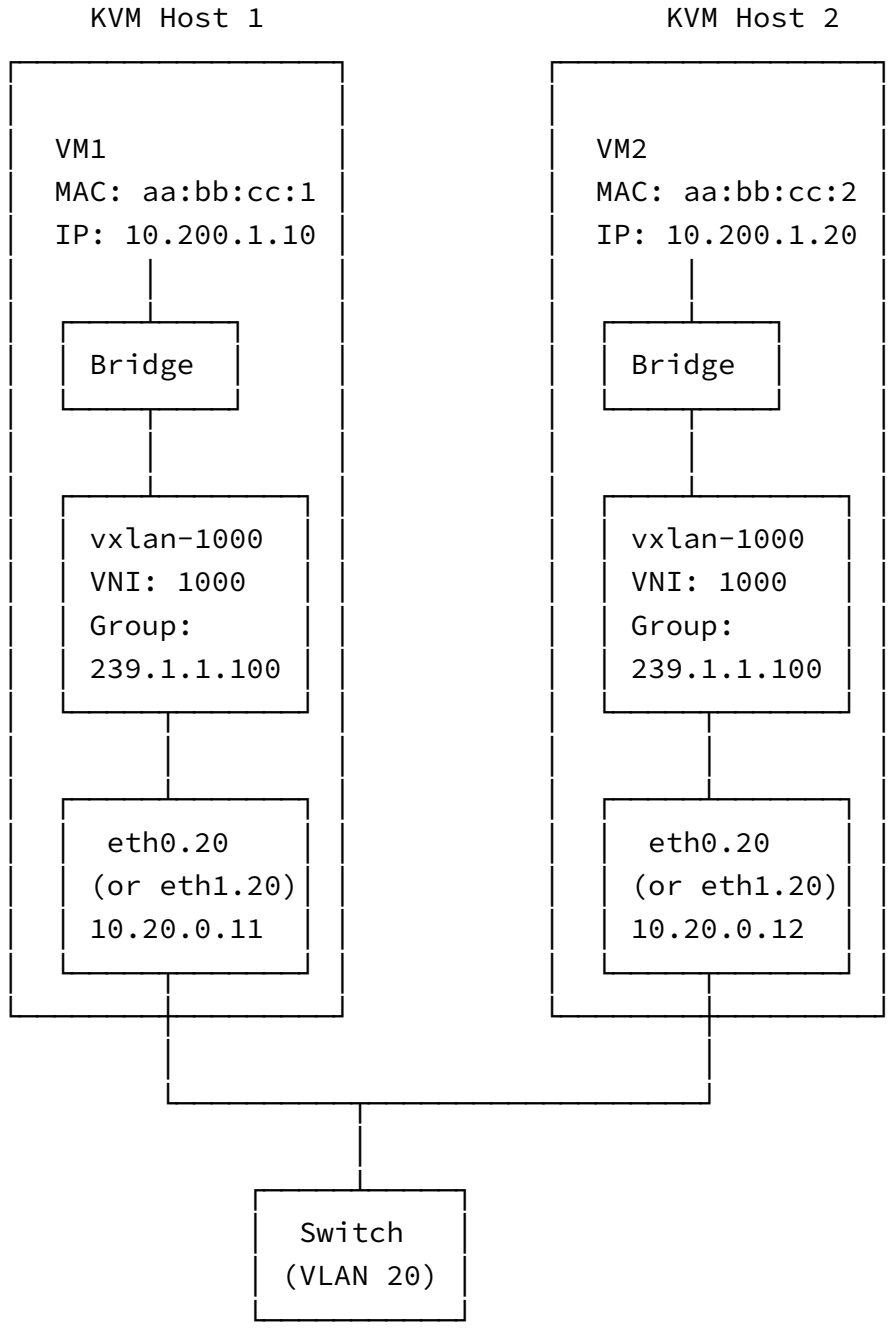
5.6 VXLAN Customer Network Architecture

5.6.1 Host-Level VXLAN Configuration





5.6.2 Multi-Host VXLAN Communication



Communication Flow:

1. VM1 sends to VM2 (10.200.1.20)
2. ARP: Multicast 239.1.1.100 → all VTEPs receive
3. VM2 responds, Host 1 learns MAC aa:bb:cc:2 at 10.20.0.12
4. Future frames: unicast encapsulation to 10.20.0.12

```
5. VXLAN overhead: 50 bytes (hence MTU 1450 for VMs)
```

5.6.3 VXLAN Packet Encapsulation

```
Inside VM (payload):
```

```
┌─────────────────────────────────────────────┐
│ Ethernet Header (14 bytes)                  │
│  Dest MAC: aa:bb:cc:2                       │
│  Src MAC:  aa:bb:cc:1                       │
├─────────────────────────────────────────────┤
│ IP Header (20+ bytes)                       │
│  Dest IP: 10.200.1.20                       │
│  Src IP:  10.200.1.10                       │
├─────────────────────────────────────────────┤
│ TCP/UDP + Application Data                  │
└─────────────────────────────────────────────┘
```

```
After VXLAN encapsulation (on wire):
```

```
┌─────────────────────────────────────────────┐
│ Outer Ethernet (14 bytes)                   │
│  Dest MAC: Switch MAC                       │
│  Src MAC:  Host 1 eth0.20 MAC               │
├─────────────────────────────────────────────┤
│ Outer IP (20 bytes)                         │
│  Dest IP: 10.20.0.12 (Host 2 on VLAN 20)    │
│  Src IP:  10.20.0.11 (Host 1 on VLAN 20)    │
├─────────────────────────────────────────────┤
│ Outer UDP (8 bytes)                         │
│  Dest Port: 4789 (VXLAN standard)           │
│  Src Port:  Random (flow hash)              │
├─────────────────────────────────────────────┤
│ VXLAN Header (8 bytes)                      │
│  VNI: 1000                                  │
├─────────────────────────────────────────────┤
│ Inner Ethernet + IP + Data (original frame) │
│  (Same as payload above)                    │
└─────────────────────────────────────────────┘
```

```
Total overhead: 50 bytes
VM MTU: 1500 - 50 = 1450 bytes
VXLAN transport VLAN 20 MTU: 1504 bytes (accommodates overhead)
```

Key Point: VXLAN traffic is isolated on VLAN 20, separate from management (VLAN 10) and storage (VLAN 31, 32). This provides clear traffic separation and allows independent

QoS policies.

5.7 MTU Configuration Strategy

5.7.1 Per-Network MTU Settings

```
Network Layer                          MTU Setting
═══════════════════════════════════════════════════

Physical Interfaces (Profile 1):
├─ eth0                                1504 bytes

Physical Interfaces (Profile 2+):
├─ eth0 (management)                   1500 bytes
└─ eth1 (VXLAN + storage)              9000 bytes

Bonded Interfaces (Profile 3+):
├─ bond0 (management)                  1500 bytes
└─ bond1 (VXLAN + storage)             9000 bytes

VLAN Interfaces:
├─ vlan.10 (management)                1500 bytes
├─ vlan.20 (VXLAN transport)           1504 bytes
├─ vlan.31 (storage1)                  9000 bytes
├─ vlan.32 (storage2)                  9000 bytes
└─ vlan.33+ (additional storage)       9000 bytes

VXLAN Interfaces:
└─ vxlan-* (all customer VXLANs)       1450 bytes

Bridges:
├─ br-vxlan-* (customer bridges)       1450 bytes
└─ br-storage (if used)                9000 bytes

VM/Container Interfaces:
├─ Customer network VMs                1450 bytes
└─ Storage network VMs                 9000 bytes
```

5.7.2 Rationale

VLAN 20 (VXLAN transport): 1504 Accommodates VXLAN overhead (50 bytes) while keeping customer VM MTU at 1450

VLAN 10 (Management): 1500 Standard MTU, no special requirements

Storage VLANs (31, 32+): 9000 Jumbo frames improve iSCSI performance (10-15% throughput gain, reduced CPU)

VXLAN interfaces: 1450 Leaves headroom for encapsulation without fragmentation

Customer VMs: 1450 Standard for internet connectivity, prevents fragmentation issues

---

5.8 Switch Configuration Requirements

5.8.1 Minimum Requirements (All Stages)

VLAN Support:

- Trunk port configuration
- Multiple VLANs per port (10, 20, 31, 32 minimum)
- Native VLAN handling (recommend unused VLAN for security)

Frame Size:

- Support for 1522+ byte frames (VLAN tagged)
- Support for 1608+ byte frames (VXLAN on VLAN 20)
- Jumbo frame support on storage ports (9000+ bytes)

5.8.2 Profile 2+ Requirements

IGMP Snooping (recommended for VXLAN efficiency on VLAN 20):

Purpose Intelligent multicast forwarding

Without IGMP snooping Multicast flooded to all ports (works but inefficient)

With IGMP snooping Multicast only to interested ports

Configuration concept (vendor agnostic):

- Enable IGMP snooping globally
- Enable on VLAN 20 (VXLAN transport network)
- Optionally enable IGMP querier if no router present

Port Configuration (management network example):

```
Interface: Port connecting to host management NIC (eth0)
Mode: Trunk
Allowed VLANs: 10
Native VLAN: None (or unused VLAN 999)
Allow untagged: Yes (for bootstrap only, can be removed after)
MTU: 1600
```

Port Configuration (data network example):

```
Interface: Port connecting to host data NIC (eth1)
Mode: Trunk
Allowed VLANs: 20, 31, 32
MTU: 9216 (accommodates jumbo frames + overhead)
IGMP Snooping: Enabled (for VLAN 20)
Spanning Tree: Edge port (PortFast equivalent)
```

Port Configuration (storage host example):

```
Interface: Port connecting to storage host
Mode: Trunk
Allowed VLANs: 10, 31 (or 32, 33, etc.)
MTU: 9216
```

5.8.3 Profile 4 Base Requirements (Standard LACP)

Link Aggregation (LACP/802.3ad):

Purpose Bonding multiple links for bandwidth aggregation
Requirement Standard LACP support on managed switches

Configuration notes:

- Each switch independently configured
- No coordination between switches required
- Standard feature on managed switches
- Each bond member connects to different switch

LAG Configuration Concept (per switch):

```
Port-Channel/LAG: lag1
Members: port1 (from this switch only)
Mode: LACP Active
Load Balance: src-dst-ip-port
Allowed VLANs: 10, 20, 31, 32
MTU: 9216

Note: Each switch has its own LAG config
Host bond spans both switches independently
```

5.8.4 Vendor-Agnostic Configuration Checklist

For all switches supporting HPS:

- ☐ VLAN support with trunk ports
- ☐ MTU 1522+ on management ports
- ☐ MTU 1608+ on VXLAN transport ports (VLAN 20)
- ☐ MTU 9216+ on storage network ports (VLAN 31+)

- ☐ IGMP snooping available and enabled on VLAN 20
- ☐ Spanning tree configured (edge ports for host connections)
- ☐ Allow untagged traffic for bootstrap (can be removed after deployment)

For Profile 4 base deployments:

- ☐ LACP/802.3ad support (standard on managed switches)
- ☐ Per-switch LAG configuration
- ☐ No coordination between switches needed

For Profile 4 with MC-LAG enhancement (optional):

- ☐ Switch stacking OR MC-LAG capability
- ☐ Peer link configuration between switches
- ☐ Coordinated LAG across switch pair

5.9 Essential Configuration Commands

5.9.1 Bootstrap: iPXE VLAN Transition Script

Served to host during initial PXE boot:

```
#!ipxe
# HPS Bootstrap Script - Transition to VLAN 10

echo ========================================
echo HPS Network Bootstrap
echo ========================================

echo Configuring management VLAN 10
vcreate --tag 10 net0 || goto failed

echo Requesting IP address on VLAN 10
dhcp net0-10 || goto failed

echo Management network configured
echo IP: ${net0-10/ip}
echo Gateway: ${net0-10/gateway}

echo Chainloading main boot manager
chain http://${next-server}/cgi-bin/boot_manager.sh?phase=main ||
 ↪  goto failed

:failed
echo Bootstrap failed - dropping to iPXE shell
shell
```

5.9.2 Host-Level VXLAN Setup

Create VXLAN interface with multicast on VLAN 20:

```
ip link add vxlan-cust-a type vxlan \
  id 1000 \
  dstport 4789 \
  group 239.1.1.100 \
  dev eth0.20 \
  ttl 5 \
  learning

# Note: dev eth0.20 (or eth1.20) - VXLAN on dedicated VLAN 20
```

Create bridge and attach VXLAN:

```
ip link add br-vxlan-cust-a type bridge
ip link set br-vxlan-cust-a type bridge stp_state 0
ip link set vxlan-cust-a master br-vxlan-cust-a
```

Set MTU and bring up:

```
ip link set vxlan-cust-a mtu 1450
ip link set br-vxlan-cust-a mtu 1450
ip link set vxlan-cust-a up
ip link set br-vxlan-cust-a up
```

Verify multicast group membership:

```
ip maddr show dev eth0.20 | grep 239.1.1.100
```

Check learned MAC addresses:

```
bridge fdb show dev vxlan-cust-a
```

5.9.3 Bonding Configuration

Active-backup bond (Profile 3):

```
ip link add bond0 type bond mode active-backup
ip link set bond0 type bond miimon 100 primary eth0
ip link set eth0 master bond0
ip link set eth1 master bond0
ip link set bond0 up
```

LACP bond (Profile 4 base):

```
ip link add bond0 type bond mode 802.3ad
ip link set bond0 type bond miimon 100 lacp_rate fast
 ↪  xmit_hash_policy layer3+4
ip link set eth0 master bond0
ip link set eth1 master bond0
ip link set bond0 up
```

5.10 Network Component Summary

The following components require configuration management functions. Each represents a distinct functional area for automation:

5.10.1 1. Bootstrap Manager

Purpose: Handle diskless PXE boot and rapid VLAN transition

Responsibilities:

- Serve DHCP responses for untagged bootstrap requests
- Provide iPXE bootloader binary
- Generate iPXE VLAN configuration script
- Detect bootstrap vs runtime phase
- Serve appropriate boot files per phase
- Log bootstrap attempts and transitions

Key considerations:

- Untagged phase exposure (~3-7 seconds)
- VLAN 10 transition via iPXE
- Security mode handling (exploratory/production/high)
- Fallback mechanisms if iPXE fails

5.10.2 2. Physical Interface Management

Purpose: Detect, configure, and manage physical NICs (eth0, eth1, etc.)

Responsibilities:

- Enumerate available physical interfaces
- Detect link speed and status (1G vs 10G)
- Set MTU on physical interfaces based on purpose
- Configure promiscuous mode if needed
- Handle interface state (up/down)
- Identify management vs data interfaces

Key considerations:

- Profile detection (how many NICs available?)
- 1G management, 10G data differentiation
- Validation of physical connectivity
- MTU requirements per profile and purpose

5.10.3 3. VLAN Interface Management

Purpose: Create and manage 802.1Q VLAN interfaces on physical or bonded interfaces

Responsibilities:

- Create VLAN interfaces (vlan.10, vlan.20, vlan.31, vlan.32+)
- Map VLANs to physical interfaces based on deployment profile
- Set MTU on VLAN interfaces (1500 mgmt, 1504 VXLAN, 9000 storage)
- Handle VLAN interface lifecycle
- Validate VLAN tag configuration

Key considerations:

- Profile-specific VLAN-to-interface mapping
- Infrastructure VLANs (10, 20, 31, 32+) are predefined
- VLAN 10: Management
- VLAN 20: VXLAN transport (dedicated)
- VLAN 31+: Storage networks (one per storage host)
- MTU inheritance and override per VLAN purpose

5.10.4 4. Bonding/Link Aggregation Management

Purpose: Create and manage bonded interfaces for redundancy and bandwidth aggregation

Responsibilities:

- Create bond interfaces (bond0, bond1)
- Configure bonding mode (active-backup, 802.3ad)
- Add/remove slave interfaces
- Set bonding parameters (miimon, lacp_rate, xmit_hash_policy)
- Monitor bond status and failover
- Handle primary interface selection

Key considerations:

- Profile 3 uses active-backup (no switch dependency)
- Profile 4 uses 802.3ad/LACP (standard managed switch feature)
- Separate bonds for management and data
- Failover detection and recovery
- MC-LAG is optional enhancement (not required)

5.10.5 5. Bridge Management

Purpose: Create and manage Linux bridges for VM/container attachment

Responsibilities:

- Create bridge interfaces
- Configure bridge parameters (STP state, filtering)
- Attach interfaces to bridges (VLAN, VXLAN)
- Set MTU on bridges
- Manage bridge FDB (forwarding database)

Key considerations:

- Bridges for VXLAN customer networks
- Optional bridges for storage networks
- STP disabled for VXLAN bridges (no loops with VXLAN)
- MTU matching underlying interfaces

5.10.6 6. VXLAN Interface Management

Purpose: Create and manage VXLAN tunnel endpoints (VTEPs)

Responsibilities:

- Create VXLAN interfaces with VNI assignment
- Configure multicast groups
- Set VXLAN parameters (dstport, ttl, learning)
- Attach VXLAN to VLAN 20 transport interface
- Set MTU (1450) on VXLAN interfaces
- Monitor VXLAN FDB and remote VTEPs

Key considerations:

- VNI allocation per customer network
- Multicast group assignment (one per VNI)
- Transport over VLAN 20 (dedicated VXLAN transport VLAN)
- VLAN 20 on appropriate physical interface (eth0.20, eth1.20, or bond.20)
- MAC learning enabled for automatic peer discovery
- Isolation from management and storage traffic

5.10.7 7. IP Address Management

Purpose: Assign and manage IP addresses on interfaces

Responsibilities:

- Assign static IPs to infrastructure interfaces
- Configure subnet masks and network parameters
- Set default routes if needed

- Handle IP address conflicts
- Manage IP address pools for allocation
- Track bootstrap vs runtime IP assignments

Key considerations:

- Bootstrap network: 192.168.100.0/24 (untagged, temporary)
- Management network: 192.168.10.0/24 (VLAN 10)
- VXLAN transport: 10.20.0.0/24 (VLAN 20)
- Storage networks: 10.31.0.0/24, 10.32.0.0/24, etc. (VLAN 31+)
- Customer networks: 10.200.0.0/16+ (VXLAN overlays)
- Per-host IP assignments from registry

5.10.8 8. Routing and Policy Configuration

Purpose: Configure routing tables and policy routing

Responsibilities:

- Set default routes (if needed)
- Configure source-based routing (if multiple networks)
- Ensure storage traffic uses correct interface
- Ensure VXLAN traffic uses VLAN 20
- Prevent unwanted cross-network routing
- Configure IP forwarding state

Key considerations:

- Typically no routing needed (all services local)
- Disable IP forwarding unless host is router/firewall
- Firewall rules to prevent customer-to-infrastructure access
- Separate management, VXLAN, and storage paths

5.10.9 9. Multicast Configuration

Purpose: Enable and configure multicast support for VXLAN

Responsibilities:

- Enable multicast reception on VLAN 20 interfaces
- Configure IGMP parameters
- Join multicast groups (automatic via VXLAN, but validate)
- Monitor multicast group membership
- Handle multicast routing settings if needed

Key considerations:

- Required for VXLAN multicast mode on VLAN 20
- IGMP version compatibility
- Multicast TTL settings
- IGMP snooping on switches (VLAN 20)

5.10.10 10. MTU Management

Purpose: Configure Maximum Transmission Unit across network stack

Responsibilities:

- Set MTU on physical interfaces based on purpose
- Propagate MTU to VLAN interfaces
- Set appropriate MTU on VXLAN interfaces (1450)
- Configure MTU on VLAN 20 (1504 for VXLAN overhead)
- Configure MTU on storage VLANs (9000 for jumbo frames)
- Validate MTU path between hosts

Key considerations:

- Profile 1: 1504 on eth0 (accommodates all traffic)
- Profile 2+: 1500 on management NIC, 9000 on data NIC
- VLAN 10 (management): 1500
- VLAN 20 (VXLAN transport): 1504
- VLAN 31+ (storage): 9000
- VXLAN interfaces: 1450 (accounting for 50-byte overhead)
- Path MTU discovery testing

5.10.11 11. Network Validation and Testing

Purpose: Verify network configuration correctness and connectivity

Responsibilities:

- Validate interface state (up/down, link speed)
- Test VLAN connectivity between hosts
- Verify VXLAN tunnel establishment on VLAN 20
- Check multicast group membership
- Test MTU end-to-end (ping with DF flag)
- Validate bridge FDB learning
- Monitor for errors/drops
- Verify bootstrap to runtime transition

Key considerations:

- Pre-deployment validation

- Bootstrap phase verification
- VLAN 10 management connectivity
- VLAN 20 VXLAN multicast functionality
- Storage VLAN (31+) jumbo frame support
- Continuous health monitoring
- Troubleshooting support
- Performance metrics collection

5.10.12 12. Network Registry/Configuration Management

Purpose: Store and manage network definitions and host assignments

Responsibilities:

- Define logical networks (management, VXLAN transport, storage)
- Map networks to VLANs (10, 20, 31+)
- Assign VNI numbers to customer networks
- Store per-host interface assignments
- Track IP address allocations
- Store multicast group assignments
- Maintain deployment profile configuration
- Track bootstrap security mode

Key considerations:

- Central source of truth for network config
- VLAN numbering scheme (10, 20, 31+)
- Storage VLAN allocation (one per storage host)
- VXLAN on dedicated VLAN 20
- Version control of network definitions
- Cluster-specific configurations
- Customer network isolation mappings
- Bootstrap mode configuration

5.10.13 13. VM/Container Network Attachment

Purpose: Attach VMs and containers to customer network bridges

Responsibilities:

- Configure libvirt/KVM network definitions
- Attach VM interfaces to specific bridges
- Create veth pairs for container attachment
- Set MAC addresses

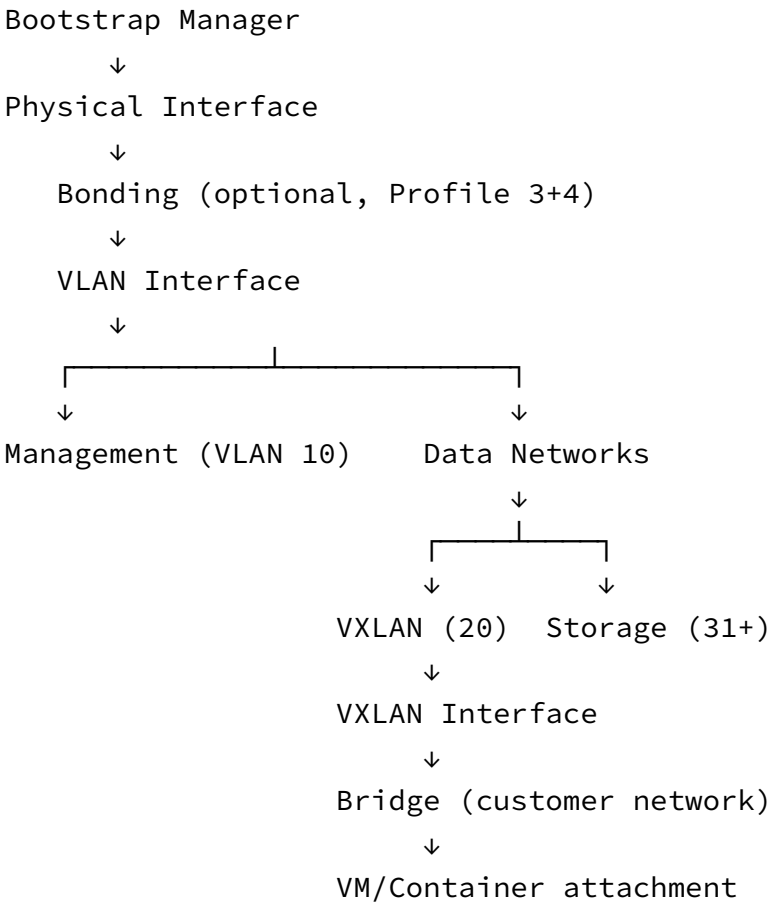
- Configure MTU on VM/container interfaces (1450 for VXLAN networks)
- Handle hot-plug/unplug if needed

Key considerations:

- Libvirt XML generation
- Docker network integration
- Customer network isolation enforcement
- MAC address management
- MTU settings (1450 for customer VMs on VXLAN)

5.11 Integration Points

These components interact as follows:



Each component should be implemented as independent, testable functions that can be composed to build the complete network stack based on the deployment profile.

5.12 Deployment Profile Decision Matrix

Available Hardware	Recommended Profile	Management NIC
1 NIC per host Any switch	Profile 1 (Testing only) No switch config	Shared 10G Bootstrap: untagged Runtime: VLAN 10
1G + 10G NIC per host 1 managed switch	Profile 2 (Production) Configure VLANs IGMP snooping	Dedicated 1G Bootstrap: untagged Runtime: VLAN 10
1G + 10G NIC per host 2 independent switches	Profile 3 (HA) Configure VLANs Optional bonding	Dedicated 1G Optional bonding Bootstrap: untagged
2x1G + 2x10G per host 2 switches with LACP	Profile 4 (Max perf) LACP bonding VLANs + LAG	Bonded 2x1G Bootstrap: untagged Runtime: bond.10

5.12.1 VLAN Usage Summary by Profile

Profile 1 (Single 10G NIC):

- VLAN 10: Management (eth0.10)
- VLAN 20: VXLAN transport (eth0.20)
- VLAN 31: Storage 1 (eth0.31)
- VLAN 32: Storage 2 (eth0.32)

Profile 2 (1G mgmt + 10G data):

- VLAN 10: Management (eth0.10) - on 1G NIC
- VLAN 20: VXLAN transport (eth1.20) - on 10G NIC
- VLAN 31: Storage 1 (eth1.31) - on 10G NIC
- VLAN 32: Storage 2 (eth1.32) - on 10G NIC

Profile 3 (with optional bonding):

- Same as Profile 2, but interfaces may be bonded
- VLAN 10: Management (bond0.10)
- VLAN 20: VXLAN transport (bond1.20)
- VLAN 31: Storage 1 (bond1.31)
- VLAN 32: Storage 2 (bond1.32)

Profile 4 (full bonding):

- VLAN 10: Management (bond0.10) - on 2x1G bonded
- VLAN 20: VXLAN transport (bond1.20) - on 2x10G bonded

- VLAN 31: Storage 1 (bond1.31) - on 2x10G bonded
- VLAN 32: Storage 2 (bond1.32) - on 2x10G bonded

Each profile builds upon the same logical network definitions (VLANs 10, 20, 31, 32+), differing only in physical interface mapping and redundancy features. This allows progressive hardware enhancement without reconfiguring the logical network layer. # Governance

5.13 AI Use Statement for HPS System Development

5.13.1 AI-Assisted Development Disclosure

The HPS System project has been developed with assistance from AI language models, specifically Large Language Models (LLMs) trained on diverse datasets. This statement provides transparency regarding the use of AI in our development process and explains our licensing decisions.

5.13.2 Use of AI in Development

AI language models have been utilized throughout the development of HPS System to:

- Generate code implementations for system administration functions
- Provide technical guidance on Linux system administration best practices
- Assist in creating documentation and code comments
- Help structure the project architecture and design patterns
- Debug and optimize bash scripts and system configurations

All AI-generated content has been reviewed, tested, and modified by human developers to ensure functionality, security, and adherence to project requirements.

5.13.3 Training Data Uncertainty

We acknowledge that the AI models used in development were trained on vast amounts of publicly available data, potentially including:

- Open source code from various licenses (GPL, MIT, Apache, BSD, etc.)
- Technical documentation and tutorials
- Stack Overflow and other technical forum discussions
- Academic papers and technical specifications
- Proprietary code that may have been inadvertently included in training datasets

Critical Point: We cannot definitively know what specific code or content was included in the AI's training data, nor can we trace the provenance of any particular suggestion or implementation pattern.

5.13.4 AGPL Licensing Decision

Given the uncertainty around training data sources and to honor the most restrictive Free/Libre and Open Source Software (FLOSS) licenses that may have influenced the AI's outputs, we have chosen to license HPS System under the **GNU Affero General Public License v3.0 (AGPL-3.0)**.

This decision ensures:

1. **Maximum compatibility** with restrictive copyleft licenses that may have been in the training data
2. **Protection of user freedoms** by ensuring all modifications remain open source
3. **Network use compliance** requiring source disclosure even for network services
4. **Respect for the FLOSS community** by choosing the most protective common license

5.13.5 Ethical Considerations

We believe in:

- **Transparency**: Being open about our use of AI assistance
- **Attribution**: While we cannot attribute to specific sources, we acknowledge the collective contributions of the open source community
- **Reciprocity**: Giving back to the community through our own AGPL-licensed contributions
- **Continuous improvement**: Reviewing and improving our practices as AI and licensing landscapes evolve

5.13.6 Future Commitments

As the project evolves, we commit to:

- Maintaining transparency about AI use in development
- Staying informed about evolving best practices for AI-assisted open source development
- Contributing to discussions about ethical AI use in FLOSS projects
- Updating this statement as necessary to reflect changes in our practices or understanding

5.13.7 Contact

For questions or concerns about AI use in HPS System development, please open an issue on our GitHub repository or contact the project maintainers.

5.14 Important Notice on ZFS Build Scripts and Licensing

This project includes scripts that automate the process of building ZFS kernel modules from source on the user's machine. It is important to understand the legal implications surrounding this practice.

5.14.1 Legal Position Summary

- ZFS is licensed under the **Common Development and Distribution License (CDDL)**, while the Linux kernel uses the **GNU General Public License v2 (GPLv2)**. These licenses are incompatible, creating legal complexities related to distribution of ZFS modules for Linux.
- **Distributing build scripts alone**—without providing pre-compiled ZFS binaries—is generally considered **low legal risk**. This is because the build and combination of CDDL-licensed ZFS code with the GPL-licensed Linux kernel are performed locally by the user, for their personal or internal use.
- Many Linux distributions and projects provide scripts or package build files for users to compile ZFS modules themselves, thereby avoiding direct distribution of potentially legally problematic binaries.
- The **legal risk increases** if pre-compiled ZFS kernel modules are distributed, as this may be considered distribution of a combined work violating license terms.

5.14.2 User Responsibility

Users are responsible for building the software themselves on their own systems, and for ensuring compliance with all applicable licenses and legal requirements regarding ZFS usage.

6 HPS system documentation plan

6.1 Overview

- **Project introduction** – Purpose of HPS, architecture, intended use cases.
- **Design decisions** – Records of key technical choices and rationale.
- **System components** – Description of major directories and modules.

6.2 Quick start

- **Prerequisites** – Required host operating system, packages, and network setup.
- **Installation** – Deploying `hps-container` with `hps-system` and `hps-config`.
- **First cluster setup** – Initialising a cluster with `cluster-configure.sh`.
- **Booting a host** – PXE boot process and selecting a host profile.
- **Verification** – Checking service status and logs.

6.3 System administration

- **Directory layout** – Locations for configuration, logs, distributions, and packages.
- **Cluster management** – Creating, switching, and editing clusters.
- **Host management** – Adding, removing, and updating host configurations.
- **Distribution and repository management** – Managing ISOs, PXE trees, and package repositories.
- **Service management** – Starting, stopping, and reloading dnsmasq, nginx, and supervisord.
- **Backup and restore** – Protecting and recovering configuration and repository data.

6.4 Functions reference

- Automatically generated from `lib/functions.d/` and other libraries.
- One function per page with purpose, arguments, usage examples, and related functions.

6.5 Advanced configuration

- **Kickstart and preseed templates** – Structure, variables, and customisation.
- **iPXE menus** – How menus are built and extended.
- **Storage provisioning** – SCH node disk detection, reporting, and configuration.
- **Integration points** – Hooks for OpenSVC, monitoring, and external systems.

6.6 Troubleshooting

- **PXE boot issues** – Common causes and fixes.
- **Service failures** – Diagnosing and restarting services.
- **Distribution and repository problems** – Checksums, GPG keys, and synchronisation errors.
- **Network problems** – DHCP conflicts, VLAN configuration, and firewall blocks.

6.7 Development

- **Code layout** – File structure and conventions for scripts and libraries.
- **Adding functions** – Naming, argument handling, logging, and documentation guidelines.
- **Testing** – Using `cli/test.sh` and other test harnesses.
- **Contributing** – Workflow, coding standards, and submission process.

6.8 Appendices

- **Glossary** – Definitions of terms and acronyms used in HPS.
- **Environment variables** – Description of exported variables and their use.
- **Decision records** – Full list of design decisions.
- **Reference configurations** – Example cluster, host, and service configuration files.

7 Storage Disaster Recovery Runbook

7.1 Purpose

This runbook provides example step-by-step procedures for recovering from storage failures in a ZFS/iSCSI/MD RAID environment. Follow procedures carefully and in order.

7.2 Pre-Incident Preparation

7.2.1 Required Information

Document and keep updated:

```
# Storage server details
STORAGE_IP=192.168.125.116
STORAGE_POOL=storage

# KVM host details
KVM_HOST_IP=192.168.125.x
KVM_HOST_NAME=s01

# VM details
VM_NAME=sjm-explore
VM_BOOT_DISK=/dev/sda (iSCSI: lun01)
VM_DATA_DISK=/dev/vdb (MD RAID: md1)

# RAID configuration
RAID_DEVICE=/dev/md1
RAID_MEMBER_1=/dev/sdb (iqn:test01)
RAID_MEMBER_2=/dev/sdc (iqn:test02)
```

7.2.2 Monitoring Commands

Keep these commands ready:

```
# On storage server
zpool status storage
zpool iostat -v storage 2
sudo targetcli ls
```

```
# On KVM host
cat /proc/mdstat
sudo mdadm --detail /dev/md1
sudo iscsiadm -m session
lsblk -o NAME,SIZE,TYPE,MOUNTPOINT

# System-wide
dmesg -T | tail -50
sudo systemctl status target
```

7.3 Scenario 1: Single iSCSI Target Failure

7.3.1 Symptoms

- MD RAID shows degraded state [U_]
- One device in RAID marked as faulty or removed
- dmesg shows I/O errors on specific device
- VM continues operation (may be slightly slower)

7.3.2 Diagnosis

On KVM host:

```
# Check RAID status
cat /proc/mdstat
sudo mdadm --detail /dev/md1

# Identify failed device
dmesg | grep -i "error" | tail -20

# Check iSCSI sessions
sudo iscsiadm -m session -P 3 | grep -E "Target:|State:|Attached"
```

On storage server:

```
# Check ZFS pool health
zpool status storage

# Check target status
sudo targetcli ls /iscsi

# Verify zvol accessibility
ls -lh /dev/zvol/storage/iscsi-*
```

7.3.3 Recovery Procedure

7.3.3.1 Step 1: Identify Root Cause

If ZFS error:

```
# Check pool errors
zpool status -v storage

# Clear transient errors (if applicable)
sudo zpool clear storage

# Scrub if needed
sudo zpool scrub storage
```

If zvol missing:

```
# Check volmode
zfs get volmode storage/iscsi-<identifier>

# Fix if needed
sudo zfs set volmode=dev storage/iscsi-<identifier>

# Trigger device creation
sudo udevadm trigger --subsystem-match=block

# Verify
ls -lh /dev/zvol/storage/iscsi-<identifier>
```

If target configuration issue:

```
# Recreate backstore
sudo targetcli

/> backstores/block create name=<identifier> \
   dev=/dev/zvol/storage/iscsi-<identifier> readonly=False

/> iscsi/<target-iqn>/tpg1/luns create
 ↪  /backstores/block/<identifier>

/> exit
```

7.3.3.2 Step 2: Reconnect on KVM Host

```
# Logout from specific target
sudo iscsiadm -m node -T <target-iqn> -u

# Wait 5 seconds
sleep 5

# Login again
sudo iscsiadm -m node -T <target-iqn> -l
```

```
# Verify device appeared
lsblk | grep 10G
dmesg | tail -10
```

7.3.3.3 Step 3: Re-add to RAID

```
# Identify which device came back (sdb, sdc, sdd, etc.)
NEW_DEVICE=/dev/sdX  # Replace X

# Test device is accessible
sudo dd if=$NEW_DEVICE of=/dev/null bs=1M count=10 iflag=direct

# If OLD superblock exists, clean it
sudo mdadm --zero-superblock $NEW_DEVICE

# Add back to array
sudo mdadm --manage /dev/md1 --add $NEW_DEVICE

# Monitor rebuild
watch cat /proc/mdstat
```

7.3.3.4 Step 4: Verify Recovery

```
# Wait for rebuild to complete
# Status should show [UU]

# Check final state
sudo mdadm --detail /dev/md1

# Verify no errors
dmesg | grep -i error | tail -20

# Test write performance
sudo dd if=/dev/zero of=/dev/md1 bs=1M count=100 oflag=direct
```

7.3.4 Expected Timeline

- Detection: Immediate to 2 minutes
- Diagnosis: 2-5 minutes
- Repair: 5-10 minutes
- Rebuild (10GB): 5-10 minutes
- **Total: 15-30 minutes**

7.4 Scenario 2: Complete Storage Server Network Failure

7.4.1 Symptoms

- All iSCSI sessions disconnect
- Multiple/all RAID arrays fail
- VMs freeze or crash if boot disks affected
- Network unreachable errors

7.4.2 Diagnosis

On KVM host:

```
# Check iSCSI sessions
sudo iscsiadm -m session
# May show no sessions

# Test network connectivity
ping $STORAGE_IP

# Check RAID status
cat /proc/mdstat
# May show [__] or stopped

# Check VM status
sudo virsh list --all
```

7.4.3 Recovery Procedure

7.4.3.1 Step 1: Restore Network

Verify physical connectivity:

- Check cables
- Check switch ports
- Verify network interface up on storage server

On storage server:

```
# Check network interface
ip addr show
sudo systemctl status networking

# Test from storage server to KVM host
ping $KVM_HOST_IP

# Verify iSCSI service listening
sudo ss -tlnp | grep 3260
```

7.4.3.2 Step 2: Reconnect iSCSI Sessions

On KVM host:

```
# Attempt to reconnect all targets
sudo iscsiadm -m node -l

# If that fails, try manual reconnection
sudo iscsiadm -m discovery -t st -p $STORAGE_IP
sudo iscsiadm -m node -l

# Verify sessions restored
sudo iscsiadm -m session
```

7.4.3.3 Step 3: Assess RAID Damage

```
# Check all RAID arrays
cat /proc/mdstat

# For each degraded array:
sudo mdadm --detail /dev/mdX

# Identify missing devices
lsblk -o NAME,SIZE,TYPE
```

7.4.3.4 Step 4: Rebuild RAIDs

For each RAID array:

```
# If RAID is stopped, try to assemble
sudo mdadm --assemble /dev/md1 /dev/sdX /dev/sdY

# If only one device missing, add it back
sudo mdadm --manage /dev/md1 --add /dev/sdX

# Monitor rebuild
watch cat /proc/mdstat
```

7.4.3.5 Step 5: Recover VMs

If VMs are down:

```
# Check VM state
sudo virsh list --all

# Attempt to start
sudo virsh start $VM_NAME

# Monitor logs
sudo virsh console $VM_NAME
```

```
# Or
tail -f /var/log/libvirt/qemu/$VM_NAME.log
```

7.4.4 Expected Timeline

- Network restoration: 5-30 minutes (depends on root cause)
- iSCSI reconnection: 2-5 minutes
- RAID assessment: 5-10 minutes
- RAID rebuild: 30-120 minutes (depends on size)
- VM recovery: 5-15 minutes
- **Total: 1-3 hours**

7.5 Scenario 3: ZFS Pool Degraded or Failed

7.5.1 Symptoms

- `zpool status` shows DEGRADED or FAULTED
- Disk errors in dmesg
- I/O performance severely degraded
- iSCSI targets may become read-only or unavailable

7.5.2 Diagnosis

```
# On storage server
zpool status -v storage

# Check for disk errors
dmesg | grep -i "ata\|scsi\|error"

# Check SMART status
sudo smartctl -a /dev/disk/by-id/<disk-id>

# Verify both mirror members
ls -lh /dev/disk/by-id/ | grep ST12000
```

7.5.3 Recovery Procedure - Mirror Member Failure

7.5.3.1 Step 1: Identify Failed Disk

```
# Check pool status
zpool status storage

# Note which disk shows FAULTED or UNAVAIL
# Example: ata-ST12000VN0008-2YS101_ZRT2KAHX
```

7.5.3.2 Step 2: Replace Disk (Hot Swap if Supported)

Physical replacement:

1. Identify physical disk bay
2. Power down if hot-swap not supported
3. Replace failed disk
4. Power on / wait for disk detection

7.5.3.3 Step 3: Resilver New Disk

```
# Replace in pool
sudo zpool replace storage \
  ata-ST12000VN0008-2YS101_OLDSERIAL \
  ata-ST12000VN0008-2YS101_NEWSERIAL

# Monitor resilver
watch zpool status storage

# Check progress
zpool status -v storage | grep -i resilver
```

7.5.3.4 Step 4: Verify Pool Health

```
# After resilver completes
zpool status storage
# Should show ONLINE

# Scrub to verify
sudo zpool scrub storage

# Monitor scrub
watch zpool status storage
```

7.5.4 Expected Timeline

- Diagnosis: 5-10 minutes
- Physical replacement: 15-60 minutes
- Resilver (12TB): 6-12 hours
- Scrub: 4-8 hours
- **Total: 12-24 hours**

7.6 Scenario 4: Corrupted MD RAID Metadata

7.6.1 Symptoms

- `mdadm --detail` fails with “cannot load array metadata”
- RAID shows as “broken” or “inactive”
- Devices exist but RAID won’t recognize them
- State shows FAILED despite devices being present

7.6.2 Diagnosis

```
# Check RAID status
cat /proc/mdstat
sudo mdadm --detail /dev/md1

# Examine individual devices
sudo mdadm --examine /dev/sdb
sudo mdadm --examine /dev/sdc

# Check for UUID mismatches
```

7.6.3 Recovery Procedure

⚠ WARNING: This procedure risks data loss. Only proceed if: - You have backups - Data is non-critical (test environment) - All other options exhausted

7.6.3.1 Step 1: Attempt Metadata Repair

```
# Stop the array
sudo umount /mnt/raid
sudo mdadm --stop /dev/md1

# Try to reassemble with --force
sudo mdadm --assemble --force /dev/md1 /dev/sdb /dev/sdc

# If successful, verify
sudo mdadm --detail /dev/md1
```

7.6.3.2 Step 2: If Repair Fails - Rebuild from One Good Member

```
# Identify which device has valid data
sudo mdadm --examine /dev/sdb
sudo mdadm --examine /dev/sdc

# Stop array
sudo mdadm --stop /dev/md1
```

```
# Assemble with single device
sudo mdadm --assemble /dev/md1 /dev/sdb --run

# Verify data is accessible
sudo fsck -n /dev/md1
sudo mount -o ro /dev/md1 /mnt/test

# If data looks good:
sudo umount /mnt/test

# Clean the second device
sudo wipefs -a /dev/sdc

# Add it back
sudo mdadm --manage /dev/md1 --add /dev/sdc

# Monitor rebuild
watch cat /proc/mdstat
```

7.6.3.3 Step 3: If Both Devices Corrupted - Complete Rebuild

```
# ⚠ THIS DESTROYS ALL DATA

# Stop array
sudo mdadm --stop /dev/md1

# Clean both devices
sudo wipefs -a /dev/sdb
sudo wipefs -a /dev/sdc

# Recreate array
sudo mdadm --create /dev/md1 \
  --level=1 \
  --raid-devices=2 \
  --bitmap=internal \
  /dev/sdb /dev/sdc

# Format
sudo mkfs.ext4 /dev/md1

# Restore from backup
# (Implementation depends on backup strategy)
```

7.6.4 Expected Timeline

- Diagnosis: 10-20 minutes
- Metadata repair attempt: 10-15 minutes
- Single-device reassembly: 15-30 minutes

- Rebuild: 30-120 minutes
- Complete rebuild + restore: 2-6 hours
- **Total: 1-6 hours**

7.7 Scenario 5: VM Cannot Access Storage

7.7.1 Symptoms

- VM shows I/O errors
- Applications freeze or timeout
- Files become inaccessible
- Filesystem goes read-only

7.7.2 Diagnosis

Inside VM:

```
# Check filesystem status
mount | grep vdb
df -h

# Check for errors
dmesg | grep -i "error\|i/o"

# Test device access
sudo dd if=/dev/vdb of=/dev/null bs=1M count=10 iflag=direct
```

On KVM host:

```
# Check RAID status
cat /proc/mdstat
sudo mdadm --detail /dev/md1

# Verify disk attachment to VM
sudo virsh domblklist $VM_NAME

# Check libvirt logs
tail -f /var/log/libvirt/qemu/$VM_NAME.log
```

7.7.3 Recovery Procedure

7.7.3.1 Step 1: Identify Layer of Failure

Test each layer bottom-up:

```
# On KVM host - test RAID device
sudo dd if=/dev/md1 of=/dev/null bs=1M count=10 iflag=direct
```

```
# If that works, issue is VM attachment
# If that fails, issue is RAID/iSCSI
```

7.7.3.2 Step 2: If RAID/iSCSI Issue

Follow Scenario 1 or 2 procedures above.

7.7.3.3 Step 3: If VM Attachment Issue

```
# Detach disk from VM
sudo virsh detach-disk $VM_NAME vdb

# Reattach
cat > /tmp/disk.xml <<EOF
<disk type='block' device='disk'>
  <driver name='qemu' type='raw' cache='none'/>
  <source dev='/dev/md1'/>
  <target dev='vdb' bus='virtio'/>
</disk>
EOF

sudo virsh attach-device $VM_NAME /tmp/disk.xml --live
```

7.7.3.4 Step 4: Inside VM - Remount

```
# If filesystem went read-only
sudo mount -o remount,rw /mnt/raid

# Or unmount and remount
sudo umount /mnt/raid
sudo fsck /dev/vdb
sudo mount /dev/vdb /mnt/raid
```

7.7.4 Expected Timeline

- Diagnosis: 5-15 minutes
- Repair: 5-30 minutes (depending on root cause)
- **Total: 10-45 minutes**

7.8 Emergency Contacts and Escalation

7.8.1 Level 1: Self-Recovery

- Use this runbook

- Check monitoring dashboards
- Review recent changes

7.8.2 Level 2: Team Escalation

- Contact: [Storage Team]
- Escalate if: >30 minutes without progress
- Escalate if: Data loss risk identified

7.8.3 Level 3: Vendor Support

- ZFS/Storage vendor: Contact
- Hardware vendor: Contact
- Escalate if: Hardware failure suspected

7.9 Post-Incident Review

After resolving any incident, document:

1. **Timeline:**
   - When first detected
   - Steps taken
   - Time to resolution
2. **Root Cause:**
   - What failed
   - Why it failed
   - How it was detected
3. **Impact:**
   - Affected VMs
   - Downtime duration
   - Data loss (if any)
4. **Improvements:**
   - Monitoring gaps
   - Documentation updates
   - Preventive measures

7.10 Preventive Maintenance

7.10.1 Daily

```
# Check RAID status
cat /proc/mdstat

# Check iSCSI sessions
sudo iscsiadm -m session
```

7.10.2 Weekly

```
# Check ZFS pool health
zpool status storage

# Review system logs
journalctl -u target -since "1 week ago"
journalctl -u iscsid -since "1 week ago"
```

7.10.3 Monthly

```
# ZFS scrub
sudo zpool scrub storage

# SMART tests on all disks
for disk in /dev/sd{a,b}; do
  sudo smartctl -t long $disk
done

# Review performance metrics
arcstat 30 10  # Sample for 5 minutes
```

7.10.4 Quarterly

```
# Test RAID rebuild
# (In test environment only)

# Verify backup/restore procedures
# Document time to restore

# Review and update runbook
# Incorporate lessons learned
```

7.11 Appendix: Quick Reference Commands

7.11.1 Storage Server

```
# ZFS health
zpool status storage
zpool list storage

# iSCSI targets
sudo targetcli ls /iscsi
sudo iscsi-lun.sh list

# Create/remove LUNs
sudo iscsi-lun.sh create <name> <size>
sudo iscsi-lun.sh remove <name>
```

7.11.2 KVM Host

```
# RAID status
cat /proc/mdstat
sudo mdadm --detail /dev/md1

# iSCSI sessions
sudo iscsiadm -m session
sudo iscsiadm -m node -l  # Login all
sudo iscsiadm -m node -u  # Logout all

# Device info
lsblk -o NAME,SIZE,TYPE,MOUNTPOINT
```

7.11.3 Inside VM

```
# Filesystem status
df -h
mount | grep vdb

# Device testing
sudo dd if=/dev/vdb of=/dev/null bs=1M count=10 iflag=direct
```

7.11.4 Performance Monitoring

```
# Storage server
zpool iostat -v storage 2
arcstat 2

# KVM host
dstat -cdngy --disk-util --io 2
iostat -xz 2
```

7.12 Glossary

Btrfs A modern Linux file system with support for snapshots, pooling, and checksumming. Considered for HPS storage but not selected due to lack of native block device export.

CCH (Compute Cluster Host) A host profile type in HPS representing a node dedicated to compute workloads.

CIDR (Classless Inter-Domain Routing) A method for allocating IP addresses and routing, using a prefix length (e.g. /24) to indicate the network mask.

DHCP (Dynamic Host Configuration Protocol) Network protocol that automatically assigns IP addresses and other network settings to devices on a network.

DHCP interface The network interface on which HPS's DHCP service (dnsmasq) listens to respond to PXE boot and other client requests.

DHCP range The range of IP addresses available for assignment via DHCP in a given network segment.

DHCP reservation A mapping of a specific MAC address to a fixed IP address in the DHCP server configuration.

DHCP server A service that hands out IP addresses and network configuration to clients; in HPS, typically provided by dnsmasq.

DHCP subnet The network segment configuration for DHCP, usually defined by IP address and netmask/CIDR.

DHCP static lease A lease configuration mapping a specific client to a specific IP, without dynamic changes.

DHCP options Additional configuration data sent by DHCP server to clients (e.g., boot file name, domain name, DNS servers).

DHCP vendor class identifier A DHCP field that can identify the client's hardware or software type.

DHCP relay A service that forwards DHCP requests from clients in one network to a DHCP server in another network.

DHCP snooping A network switch feature that limits DHCP responses to trusted ports.

DHCP starvation An attack in which an attacker sends repeated DHCP requests to exhaust the available address pool.

DHCPDISCOVER DHCP message type sent by clients to find available DHCP servers.

DHCPOFFER DHCP message type sent by servers in response to a DHCPDISCOVER, offering an IP configuration.

DHCPREQUEST DHCP message type sent by clients to request offered configuration.

DHCPACK DHCP message type sent by the server to confirm an IP lease to the client.

dnsmasq Lightweight DNS forwarder and DHCP server used by HPS to provide PXE boot and local DNS services.

DR node (Disaster Recovery node) A dedicated node used to provide failover capability and data recovery in the event of a primary node failure.

DRH (Disaster Recovery Host) Host profile type in HPS representing a disaster recovery node.

HPS (Host Provisioning Service) The provisioning framework implemented in this project for automated PXE-based OS deployment and configuration.

HPS container The Docker container providing the HPS services.

HPS config The configuration directory structure for HPS, stored separately from the core scripts to allow upgrades without overwriting site-specific settings.

HPS system The set of Bash scripts, functions, and service configurations that implement the HPS provisioning environment.

iPXE Open-source network boot firmware supporting protocols such as HTTP, iSCSI, and PXE. Used by HPS for dynamic boot menus and provisioning.

ISO (International Organization for Standardization image file) A disk image format commonly used to distribute operating system installation media.

Kickstart Automated installation method used by Red Hat-based distributions, configured via a `.ks` file.

MAC address Media Access Control address, a unique identifier assigned to a network interface.

NFS (Network File System) Protocol for sharing files over a network, not used for boot in HPS but sometimes relevant for Linux provisioning.

PXE (Preboot eXecution Environment) Network boot framework that allows a system to boot from a network interface before an OS is installed.

PXE boot menu The interactive menu shown to PXE/iPXE clients to select a boot option.

SCH (Storage Cluster Host) Host profile type in HPS representing a node dedicated to storage services.

syslog Standardised system logging protocol used to collect logs from services.

TCH (Thin Compute Host) Host profile type in HPS representing a lightweight compute node.

TFTP (Trivial File Transfer Protocol) Simple file transfer protocol used in PXE boot to transfer bootloaders and configuration.

ZFS Advanced file system with volume management, snapshots, and data integrity features. Selected in HPS for iSCSI exports due to native zvol support.

7.13 External resources

Below are external projects, code repositories, and documentation sources that are relevant to HPS.
These provide additional background, tooling, or dependencies that are either directly used or referenced in the design.

7.13.1 Btrfs

Link: https://btrfs.readthedocs.io

Summary: Linux file system with features such as snapshots, compression, and subvolumes. Considered for HPS storage but not selected as the primary iSCSI export

filesystem.

7.13.2 dnsmasq

Link: http://www.thekelleys.org.uk/dnsmasq/doc.html

Summary: Lightweight DNS forwarder and DHCP server used in HPS to provide PXE boot services, static leases, and local DNS.

7.13.3 Docker

Link: https://docs.docker.com

Summary: Containerisation platform used to run the HPS environment (`hps-container`) in a controlled, reproducible manner.

7.13.4 iPXE

Link: https://ipxe.org

Summary: Open-source network boot firmware supporting PXE, HTTP, iSCSI, and scripting. HPS uses iPXE binaries (`ipxe.efi`, `undionly.kpxe`, `snponly.efi`) for boot menus and network installs.

7.13.5 ISO images for Rocky Linux

Link: https://download.rockylinux.org/pub/rocky/

Summary: Official Rocky Linux distribution media. HPS uses these ISOs to populate PXE boot trees and perform installations.

7.13.6 Kickstart documentation

Link: https://pykickstart.readthedocs.io

Summary: Red Hat-based distributions' automated installation system. Kickstart files are used in HPS to perform unattended OS deployments.

7.13.7 OpenSVC

Link: https://www.opensvc.com

Summary: Cluster resource manager and service orchestrator considered for integration with HPS for managing ZFS-backed iSCSI storage and service failover.

7.13.8 Pandoc

Link: https://pandoc.org

Summary: Document converter used to compile HPS Markdown documentation into PDF, HTML, and other formats.

7.13.9 PXE specification

Link: https://en.wikipedia.org/wiki/Preboot_Execution_Environment

Summary: Standard network boot process for x86 systems. HPS extends PXE with iPXE for additional protocol support and boot menu scripting.

7.13.10 Rocky Linux

Link: https://rockylinux.org

Summary: Enterprise-grade Linux distribution used as a primary OS target in HPS deployments.

7.13.11 syslog protocol (RFC 5424)

Link: https://datatracker.ietf.org/doc/html/rfc5424

Summary: Standard for message logging in IP networks. HPS services can log via syslog for centralised collection.

7.13.12 TFTP

Link: https://datatracker.ietf.org/doc/html/rfc1350

Summary: Simple file transfer protocol used to serve PXE/iPXE bootloaders and configuration files to network boot clients.

7.13.13 ZFS on Linux (OpenZFS)

Link: https://openzfs.org

Summary: Advanced file system and volume manager selected for HPS storage exports due to native block device (zvol) support, snapshots, and data integrity features.

8 Storage Performance Test Report

8.1 Executive Summary

Performance testing of ZFS-backed iSCSI storage with MD RAID mirroring demonstrates excellent cache performance and acceptable throughput for lab virtualization workloads. The L2ARC (NVMe cache) achieved 100% read hit rates during mixed workloads, while SLOG effectively handled sync write operations at 140-260 MB/s. Network bandwidth (2.5GbE) represents the primary bottleneck for sequential operations.

8.2 Infrastructure Configuration

8.2.1 Storage Server

Hardware:

- UGreen DPX2800
- CPU: Intel N100 Quad-core
- RAM: 8GB (ZFS ARC- 2.6GB allocated)
- Storage Pool: 12TB usable (2× 12TB HDD in mirror)
- SLOG Device: 18.5GB NVMe partition (nvme0n1p4)
- L2ARC Device: 93GB NVMe partition (nvme0n1p5)
- Network: 2.5GbE (~250 MB/s theoretical max)

Software:

- OS: Debian Trixie
- ZFS: Native Linux ZFS
- iSCSI Target: LIO (targetcli)

8.2.2 KVM Host

Hardware:

- CPU: AMD Ryzen 7 5825U 4550 MHz
- RAM: 32GB
- Network: 2.5GbE USB to storage server
- Storage: iSCSI-attached block devices

Software:

- OS: Debian Trixie
- iSCSI Initiator: open-iscsi
- RAID: mdadm (MD RAID1)

8.2.3 Test Virtual Machine

Configuration:

- Name: sjm-explore
- Attached Storage: /dev/vdb (10GB via MD RAID)
- Filesystem: ext4
- Direct I/O: Enabled for most tests
- OS: Debian Trixie

8.3 Test Methodology

8.3.1 Test 1: Random Write (Block Device)

Configuration:

```
Engine: fio
Pattern: randwrite
Block Size: 4KB
Jobs: 4 parallel
Direct I/O: Yes
Target: /dev/sda (iSCSI block device)
Duration: 30 seconds
```

Results:

- IOPS: 9,405
- Bandwidth: 36.7 MB/s (38.5 MB/s)
- Latency Average: 423.91 µs
- Latency 99th percentile: 1,844 µs

ZFS Observations:

- SLOG activity: 86 read ops, 174 bytes
- L2ARC: Minimal activity (bootstrap phase)
- Mirror writes: 255-323 MB/s sustained

8.3.2 Test 2: Mixed Random Read/Write (70/30)

Configuration:

```
Engine: fio
Pattern: randrw (70% read, 30% write)
Block Size: 4KB
```

```
Queue Depth: 16 (capped at 1 by psync)
Jobs: 4 parallel
Direct I/O: Yes
Target: /dev/sda
Duration: 120 seconds
```

Results:

- Read IOPS: 8,418
- Write IOPS: 3,622
- Total IOPS: 12,040
- Read Bandwidth: 32.9 MB/s
- Write Bandwidth: 14.1 MB/s
- Read Latency: 322.70 µs average
- Write Latency: 349.62 µs average

Key Observations:

- L2ARC Hit Rate: 100% (all reads served from NVMe cache)
- CPU Wait: 3-10% (low, not disk-bound)
- Disk Utilization: 20-45% (plenty of headroom)
- Network: 130-240 MB/s variable usage

arcstat Output:

```
Read Operations: 21-27K ops/sec
Demand Data Reads: 14-16K
Cache Hit Rate: 100% (ddh%)
L2ARC serving all read traffic
```

8.3.3 Test 3: Sequential Write (VM through RAID)

Configuration:

```
Engine: dd
Pattern: Sequential write
Block Size: 1MB
Direct I/O: Yes
Target: MD RAID device mounted in VM
```

Results:

- KVM Host Direct: 95.8 MB/s
- VM through virtio: 111 MB/s

SLOG Activity During VM Writes:

- SLOG grew from 16K to 371MB
- Processing 1,000-2,000 write ops/sec
- Bandwidth through SLOG: 140-258 MB/s

- Mirror receiving 80-200 ops/sec (1-2 MB/s per disk)

Interpretation:

- Sync writes hit SLOG first (fast acknowledgment)
- Data flushed to mirror in transaction groups
- SLOG provides significant write acceleration

8.4 Performance Analysis

8.4.1 Bottleneck Identification

Network Limited:

- 2.5GbE = ~250 MB/s theoretical maximum
- Sequential writes: 95-111 MB/s (38-44% of network capacity)
- Random I/O: 47 MB/s aggregate (19% of network capacity)

Storage NOT Limited:

- Disk utilization: 20-45% maximum
- IronWolf capable of: 200-250 MB/s sequential per drive
- Random IOPS capability: 180 IOPS per drive
- Both disks have significant headroom

CPU NOT Limited:

- Wait states: 3-10% only
- System CPU: 18-30% during heavy I/O
- No evidence of CPU saturation

8.4.2 Cache Effectiveness

L2ARC (Read Cache):

- Hit rate: 100% during workload
- Serving 14-16K reads/sec from NVMe
- Eliminates disk reads entirely for working set
- 93GB capacity sufficient for test workloads

SLOG (Write Cache):

- Actively absorbing sync writes
- 140-260 MB/s throughput
- Latency benefit: Sub-millisecond ack vs disk latency
- Critical for database/VM workloads

ARC (Memory Cache):

- Size: 2.6GB allocated
- Working in tandem with L2ARC

- Tiered caching strategy effective

8.5 Performance Projections

8.5.1 Current Lab Configuration

Sustainable Throughput:

- Random Read: ~8,400 IOPS
- Random Write: ~3,600 IOPS

- Sequential Read: ~150-200 MB/s (estimate)
- Sequential Write: ~100-110 MB/s
- Mixed Workload: ~12,000 IOPS total

Suitable For:

- 10-15 light VMs
- 5-8 moderate workload VMs
- 2-3 database VMs
- File servers
- Web application servers

8.5.2 Production Configuration (Dual 10GbE Servers)

Projected Performance:

- Network: 2× 10GbE = ~2,000 MB/s available per path
- Storage: NVMe backend = ~3,000 MB/s capable
- Expected bottleneck: Backend storage throughput

Estimated Throughput:

- Random Read: 100K+ IOPS (NVMe limited)
- Random Write: 50K+ IOPS (mirror write penalty)
- Sequential Read: 1,000-1,500 MB/s (dual paths + cache)
- Sequential Write: 500-800 MB/s (mirror write penalty)
- Mixed Workload: 80K-120K IOPS

Suitable For:

- 50-75 VMs (mixed workload)
- 10-15 database servers
- High-availability applications
- Production workloads with SLA requirements

8.6 Failure Recovery Performance

8.6.1 RAID Rebuild Speed

Observed:

- 10GB rebuild: ~5-10 minutes
- Speed: ~21 MB/s during active workload
- Background rebuild while VMs continue operation

Factors:

- Network bandwidth shared with workload
- Source reads from L2ARC where possible
- SLOG quiet during rebuild (no sync writes)

Production Projection:

- 100GB LUN: ~50-80 minutes
- 1TB LUN: ~8-13 hours
- Speed: ~200-500 MB/s (10GbE, less contention)

8.7 Recommendations

8.7.1 Current Lab Environment

1. Performance is Adequate:
    - 12K IOPS sufficient for lab workloads
    - Low latency (<500µs average)
    - Network is limiting factor, not storage
2. Optimization Opportunities:
    - Add second NVMe for SLOG mirroring (data protection)
    - Consider 5GbE or 10GbE upgrade if sequential workloads increase
    - Current configuration is well-balanced

8.7.2 Production Deployment

1. Network:
    - Deploy dual 10GbE per host (minimum)
    - Bonded/multipath configuration
    - Separate storage VLAN
2. Storage:
    - Dual NVMe per server for mirrored SLOG
    - Larger L2ARC (256-512GB per server)
    - Consider NVMe-only pools for critical VMs

3. Redundancy:
    - Two separate storage servers
    - RAID members from different servers
    - Geographic separation if possible
4. Monitoring:
    - Track L2ARC hit rates (maintain >85%)
    - Monitor SLOG usage patterns
    - Alert on RAID degraded states
    - Network bandwidth utilization

8.8 Conclusion

The tested configuration demonstrates excellent performance for lab/development workloads. The L2ARC achieving 100% cache hit rates proves the tiered storage strategy is highly effective. SLOG acceleration provides low-latency acknowledgment for sync writes, critical for VM and database workloads.

Network bandwidth (2.5GbE) is the primary bottleneck, not storage or compute resources. The storage subsystem has significant headroom, with disks only 20-45% utilized during heavy testing.

Production deployment with dual 10GbE and separate storage servers will scale performance by 8-10× while adding true redundancy and disaster recovery capabilities. The architecture is sound and production-ready. # ZFS/iSCSI/MD RAID Architecture Guide

8.9 Overview

This architecture provides resilient storage for KVM virtualization environments using ZFS on the storage server, iSCSI for network block storage transport, and MD RAID for host-level redundancy.

8.10 Architecture Components

8.10.1 1. Storage Server Layer

Hardware:

- 2× 12TB IronWolf drives (mirror-0)
- 1× NVMe drive partitioned for:
    - SLOG (ZIL): 18.5GB partition (nvme0n1p4)
    - L2ARC: 93GB partition (nvme0n1p5)
- 2.5GbE network interface

ZFS Configuration:

```
Pool: storage
├── mirror-0 (data vdevs)
│   ├── ata-ST12000VN0008-2YS101_ZRT2KAHX
│   └── ata-ST12000VN0008-2YS101_ZRT2K4VV
├── logs (SLOG)
│   └── nvme0n1p4
└── cache (L2ARC)
    └── nvme0n1p5
```

iSCSI Target Configuration:

- Service: targetcli (LIO)
- Protocol: iSCSI over TCP port 3260
- Authentication: Disabled for lab (enable ACLs in production)
- Backing stores: ZFS zvols

Creating iSCSI LUNs:

```
# Using management script
sudo /usr/local/bin/iscsi-lun.sh create <name> <size>

# Example
sudo iscsi-lun.sh create vm-disk01 100G
```

8.10.2 2. KVM Host Layer

Network:

- 2.5GbE connection to storage server
- IP: 192.168.125.x

iSCSI Initiator:

```
# Discovery
sudo iscsiadm -m discovery -t st -p <storage-server-ip>

# Login to specific target
sudo iscsiadm -m node -T iqn.2024-11.local.storage-01:<name> -l

# Check sessions
sudo iscsiadm -m session
```

MD RAID Configuration:

```
# Create mirror across two iSCSI LUNs
sudo mdadm --create /dev/md1 \
  --level=1 \
  --raid-devices=2 \
  --bitmap=internal \
  /dev/sdX /dev/sdY
```

```
# Format and mount
sudo mkfs.ext4 /dev/md1
sudo mount /dev/md1 /mnt/storage
```

Key Points:

- RAID members should be from DIFFERENT iSCSI targets
- Verify device sizes match before creating RAID
- Use write-intent bitmap for faster recovery

8.10.3 3. Virtual Machine Layer

Disk Attachment:

```
# Attach RAID device to VM
sudo virsh attach-disk <vm-name> /dev/md1 vdb \
  --driver qemu --subdriver raw --type block --live
```

Inside VM:

```
# Format and mount
sudo mkfs.ext4 /dev/vdb
sudo mkdir /mnt/data
sudo mount /dev/vdb /mnt/data
```

8.11 Managing Storage Outages

8.11.1 Scenario 1: Single iSCSI Target Failure

Detection:

- MD RAID transitions to degraded mode `[U_]`
- dmesg shows I/O errors
- One RAID member marked as faulty

KVM Host Response:

```
# Monitor RAID status
watch cat /proc/mdstat

# Check for failed devices
sudo mdadm --detail /dev/md1
```

Recovery Steps:

1. Fix storage server issue
2. Re-enable iSCSI target
3. Device reconnects (may have new name)
4. Add device back to RAID:

```
sudo mdadm --manage /dev/md1 --add /dev/sdX
```

5. Monitor rebuild: `watch cat /proc/mdstat`

VM Impact:

- ☒ Continues operation normally
- ☒ Write performance may degrade slightly
- ☒ No redundancy until rebuild completes

8.11.2 Scenario 2: Storage Server Network Failure

Symptoms:

- All iSCSI sessions timeout
- Multiple devices fail
- RAID may fail completely if both members affected

Prevention:

- Use multipath in production
- Separate network paths for each RAID member
- Configure appropriate iSCSI timeout values

8.11.3 Scenario 3: ZFS Volume Issues

If zvol becomes unavailable:

```
# On storage server - check volmode
zfs get volmode storage/iscsi-<name>

# Should be "dev" - if not:
sudo zfs set volmode=dev storage/iscsi-<name>

# Trigger udev
sudo udevadm trigger --subsystem-match=block

# Recreate backstore if needed
sudo targetcli backstores/block create name=<name> \
  dev=/dev/zvol/storage/iscsi-<name>
```

8.12 Critical Safety Rules

8.12.1 Device Management

DO:

- ☒ Logout specific targets: `sudo iscsiadm -m node -T <target> -u`
- ☒ Verify device sizes before RAID creation
- ☒ Use `lsblk -o NAME,SIZE,TYPE` to identify devices
- ☒ Keep VM root disks on separate LUNs from RAID members

DON'T:

- ☒ Use `iscsiadm -m session -u` (logs out ALL sessions)
- ☒ Include VM boot disks in test RAID arrays
- ☒ Stop RAID arrays while VMs are writing
- ☒ Restart storage server target service during production

8.12.2 Monitoring

Essential checks:

```
# On storage server
zpool status storage
zpool iostat -v storage 2

# On KVM host
cat /proc/mdstat
sudo mdadm --detail /dev/md1
sudo iscsiadm -m session

# Track performance
arcstat 2  # ZFS ARC statistics
dstat -cdngy --disk-util --io 2
```

8.13 Production Recommendations

1. Dual Storage Servers:
    - Mirror RAID across separate physical servers
    - Each member on different server = survive server failure
    - Requires 2× 10GbE per host for bandwidth
2. Network Redundancy:
    - Bonded interfaces (LACP)
    - Multipath iSCSI configuration
    - Separate VLANs for storage traffic
3. Monitoring:
    - Monit/Nagios for disk health
    - MD RAID status alerts
    - ZFS pool health checks
    - iSCSI session monitoring
4. SLOG Redundancy:
    - Mirror SLOG devices (2× NVMe)
    - Critical for data integrity
    - Loss of SLOG = potential data loss on power failure
5. Capacity Planning:

- L2ARC sizing: 5-10× working set size
- SLOG sizing: ~5 seconds of write traffic
- Network bandwidth: 2× expected peak load

8.14 Management Scripts

iSCSI LUN Management: Located at `/usr/local/bin/iscsi-lun.sh`

```
# Create LUN
sudo iscsi-lun.sh create <identifier> <size>

# Remove LUN
sudo iscsi-lun.sh remove <identifier>

# List all LUNs
sudo iscsi-lun.sh list
```

8.15 Troubleshooting

iSCSI connection issues:

```
# Check network connectivity
ping <storage-server>

# Verify target service running
sudo systemctl status target

# Check firewall (port 3260)
sudo ss -tlnp | grep 3260
```

RAID won't accept device:

```
# Clear stale superblock
sudo mdadm --zero-superblock /dev/sdX

# Or wipe completely
sudo wipefs -a /dev/sdX
```

Device name confusion:

```
# Use persistent device paths
ls -l /dev/disk/by-path/

# Or by-id for more stable names
ls -l /dev/disk/by-id/
```

9 HPS Network Topology Design

9.1 Objectives and Intent

9.1.1 Core Objectives

Flexible Deployment Stages Support deployment from simple single-NIC setups (testing/POC) through to production multi-NIC configurations without changing logical network definitions.

Infrastructure Isolation Separate HPS infrastructure networks (management, storage, VXLAN transport) from customer workload networks completely. Customer changes never impact HPS core systems.

Multi-Host Customer Networks Enable VMs and containers across different physical hosts to communicate on private customer networks without requiring switch VLAN configuration.

Bootstrap-Friendly Support diskless PXE boot while maintaining VLAN-based security model through rapid transition to tagged networks.

Simplicity and Robustness Minimize configuration complexity. Use standard Linux kernel features. Avoid proprietary protocols or complex control planes.

Scalability Start with minimal hardware for testing, expand to production-grade redundancy and performance without reconfiguration.

9.1.2 Design Intent

The HPS network design uses **VLAN abstraction at the infrastructure level** combined with **VXLAN overlays for customer networks**. This creates a clear separation:

HPS Infrastructure Networks Predefined VLANs (10, 20, 31, 32+) for management, VXLAN transport, and storage, mapped to physical interfaces based on deployment stage.

Customer Networks VXLAN overlays (VNI 1000+) providing isolated layer-2 domains spanning all host types (KVM, Docker, physical).

The bootstrap problem (diskless PXE requiring untagged network) is solved through a rapid transition approach:

1. Initial PXE boot on untagged network (~3-7 seconds exposure)
2. iPXE chainload configures VLAN 10 for management
3. All subsequent provisioning and runtime operations on VLAN 10

This ensures:

- Infrastructure stability (core VLANs remain unchanged)
- Customer flexibility (add/modify customer networks independently)
- Progressive enhancement (add hardware without reconfiguration)
- Minimal untagged network exposure (seconds, not minutes)

9.2 Network Architecture Overview

9.2.1 Infrastructure Networks (HPS Core)

```
┌──────────────────────────────────────────────────────────────┐
│                  HPS Infrastructure                          │
│                                                              │
│  VLAN 10: Management Network (192.168.10.0/24)               │
│    - SSH, monitoring, provisioning                           │
│    - Cluster coordination                                    │
│    - 1Gbps sufficient for Stage 2+                           │
│                                                              │
│  VLAN 20: VXLAN Transport Network (10.20.0.0/24)             │
│    - VXLAN multicast traffic                                 │
│    - Customer network encapsulation                          │
│    - Isolated from management                                │
│                                                              │
│  VLAN 31: Storage Network 1 (10.31.0.0/24)                   │
│    - iSCSI Target 1                                          │
│    - MTU 9000 (jumbo frames)                                 │
│    - 10Gbps recommended                                      │
│                                                              │
│  VLAN 32: Storage Network 2 (10.32.0.0/24)                   │
│    - iSCSI Target 2                                          │
│    - MTU 9000 (jumbo frames)                                 │
│    - 10Gbps recommended                                      │
│                                                              │
│  VLAN 33+: Additional Storage (10.33.0.0/24+)                │
│    - Scale storage capacity as needed                        │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```

9.2.2 VLAN Numbering Scheme

Reserved VLAN ranges to avoid:

- VLAN 0: Reserved (priority tagging)

- VLAN 1: Default VLAN (avoid for security)
- VLAN 1002-1005: Reserved (Cisco legacy protocols)

HPS VLAN allocation:

VLAN 10 Management (safe, commonly used)

VLAN 20 VXLAN Transport (safe)

VLAN 31-99 Storage networks (safe range, room for expansion) - 31: Storage 1 - 32: Storage 2 - 33-99: Additional storage hosts as needed

This numbering avoids all known reserved ranges and provides clear logical grouping.

9.2.3 Customer Networks (VXLAN Overlays)

```
┌──────────────────────────────────────────────────────────────┐
│                Customer Overlay Networks                     │
│        (Transported over VLAN 20 VXLAN Network)              │
│                                                              │
│  VNI 1000: Customer A Private (10.200.0.0/16)                │
│    - Multicast Group: 239.1.1.100                            │
│    - Spans: KVM hosts, Docker hosts                          │
│    - MTU 1450                                                │
│                                                              │
│  VNI 1001: Customer B Private (10.201.0.0/16)                │
│    - Multicast Group: 239.1.1.101                            │
│    - Isolated from Customer A                                │
│                                                              │
│  VNI 1002: Shared External Gateway (10.250.0.0/16)           │
│    - Multicast Group: 239.1.1.102                            │
│    - Firewall VMs provide internet access                    │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```

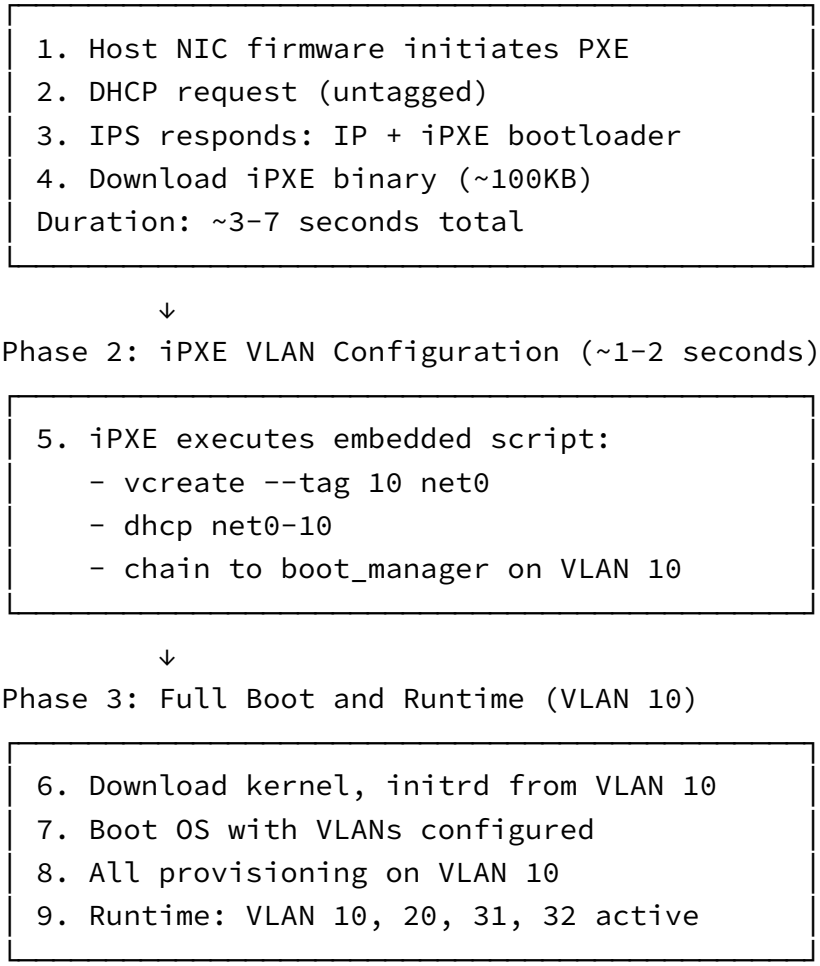
9.3 Bootstrap Process

9.3.1 The Challenge

Diskless hosts require network boot (PXE), but PXE firmware only supports untagged Ethernet. HPS runtime uses VLANs for security and isolation. This creates a bootstrap requirement.

9.3.2 HPS Solution: Rapid VLAN Transition

```
Phase 1: Initial PXE (untagged, ~3-7 seconds)
```



Untagged exposure: ~3-7 seconds

All runtime operations: VLAN-tagged

9.3.3 Bootstrap Security Modes

Exploratory Mode (Default - Stage 1) Untagged bootstrap enabled. Suitable for: Lab, testing, home, POC. Minimal security risk (seconds of exposure).

Production Mode (Stage 2+) Untagged bootstrap on dedicated 1Gb management NIC. Management traffic isolated from data networks. Suitable for: Production deployments.

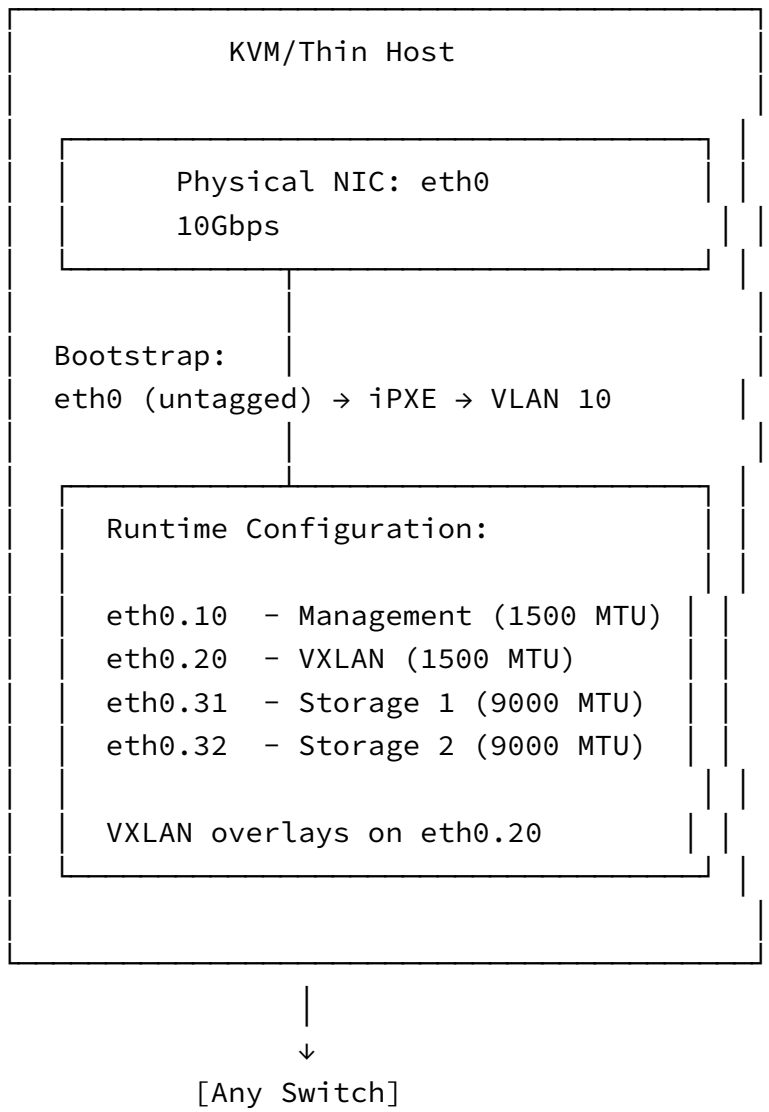
High Security Mode (Optional) No untagged bootstrap allowed. Requires: Pre-configured switch with VLAN 10 as native VLAN or out-of-band management (IPMI) for initial configuration. Suitable for: Highly regulated environments.

9.4 Deployment Stages

9.4.1 Stage 1: Single NIC (Testing/POC)

Hardware: 1 x 10G NIC per host, any switch (managed or unmanaged)

Purpose: Lab environments, home use, initial testing, proof of concept



Characteristics:

- All VLANs share single 10G interface
- Bootstrap: untagged (~3-7 seconds) → VLAN 10 transition
- Storage MTU 9000 (jumbo frames) on eth0.31, eth0.32
- Management/VXLAN MTU 1500 on eth0.10, eth0.20
- VXLAN multicast on eth0.20 (dedicated VLAN)
- Switch can be unmanaged for testing (inefficient but works)
- **Not performant - all traffic shares 10G - testing only**

Switch Requirements:

- None for unmanaged switch

- If managed: Trunk port with VLANs 10, 20, 31, 32 + allow untagged (bootstrap)

Bootstrap Process:

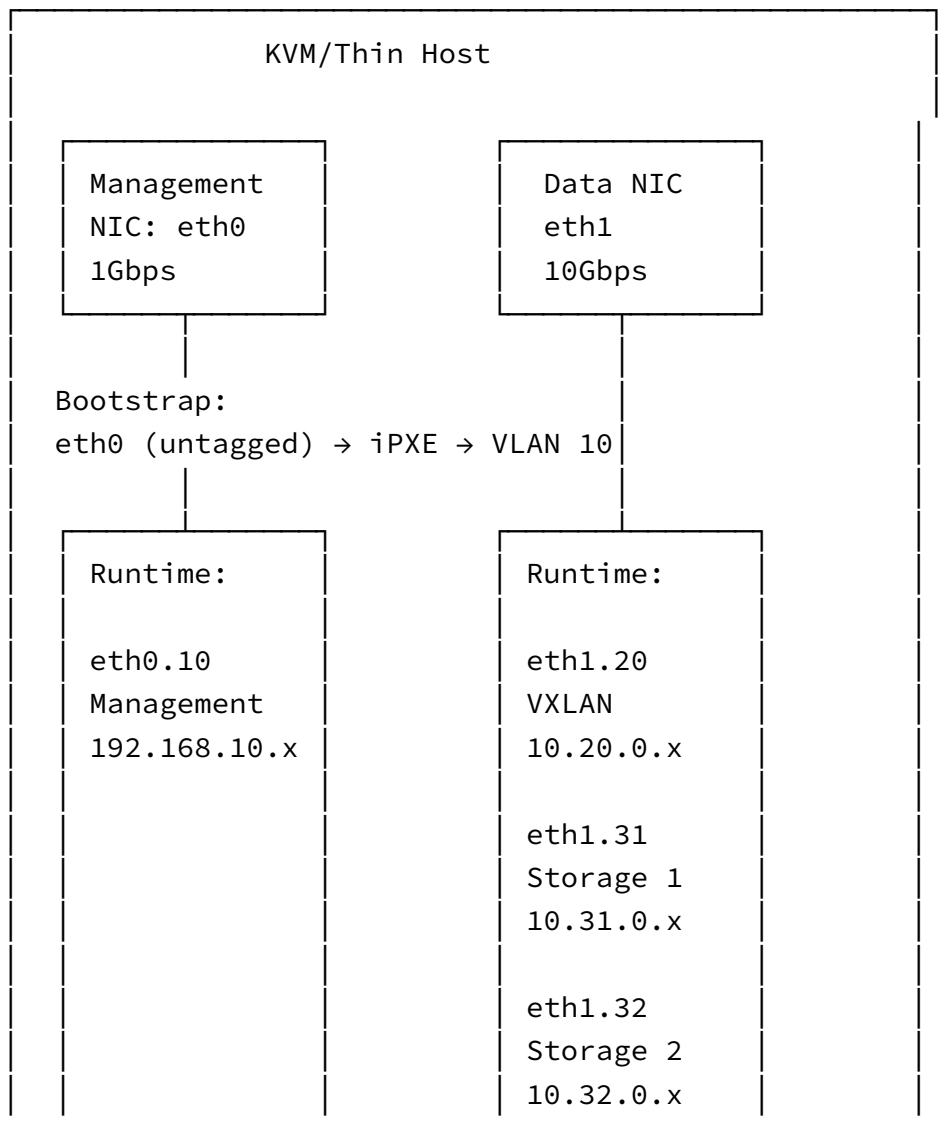
1. PXE boot (untagged) → iPXE
2. iPXE creates VLAN 10, gets IP
3. Downloads OS from VLAN 10
4. OS configures all VLANs (10, 20, 31, 32)

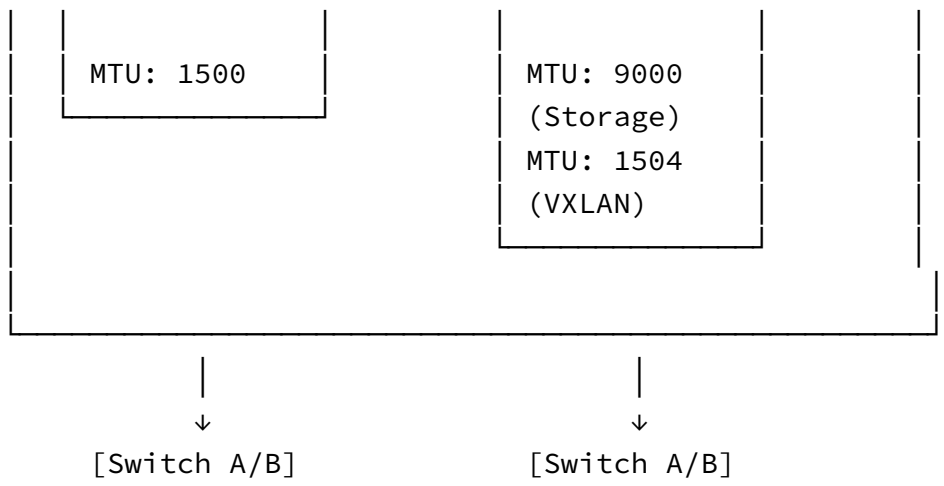
9.4.2 Stage 2: Dual NIC (Production Entry)

Hardware:

- 1 x 1G NIC (management)
- 1 x 10G NIC (VXLAN + storage)
- 1-2 managed switches

Purpose: Production deployments with network isolation





Physical Interface Assignment:

eth0 (1Gb) Management only (VLAN 10). SSH, monitoring, provisioning. Low bandwidth requirements. Dedicated interface for administrative access.

eth1 (10Gb) Data networks (VLAN 20, 31, 32). VXLAN transport (VLAN 20). Storage networks (VLAN 31, 32) with jumbo frames. High bandwidth for data operations.

Characteristics:

- Management isolated on dedicated 1Gb NIC
- VXLAN and storage share 10Gb data NIC
- Bootstrap on eth0 (untagged → VLAN 10 transition)
- Storage networks: MTU 9000 (jumbo frames for iSCSI)
- VXLAN network: MTU 1504 (accommodates VXLAN overhead)
- Management network: MTU 1500
- Clear separation: management vs data

Switch Requirements:

- Trunk ports supporting VLANs 10, 20, 31, 32
- IGMP snooping on VLAN 20 (recommended for VXLAN efficiency)
- Jumbo frame support on data ports (MTU 9000+)
- Allow untagged on management port (bootstrap)

Bootstrap Process:

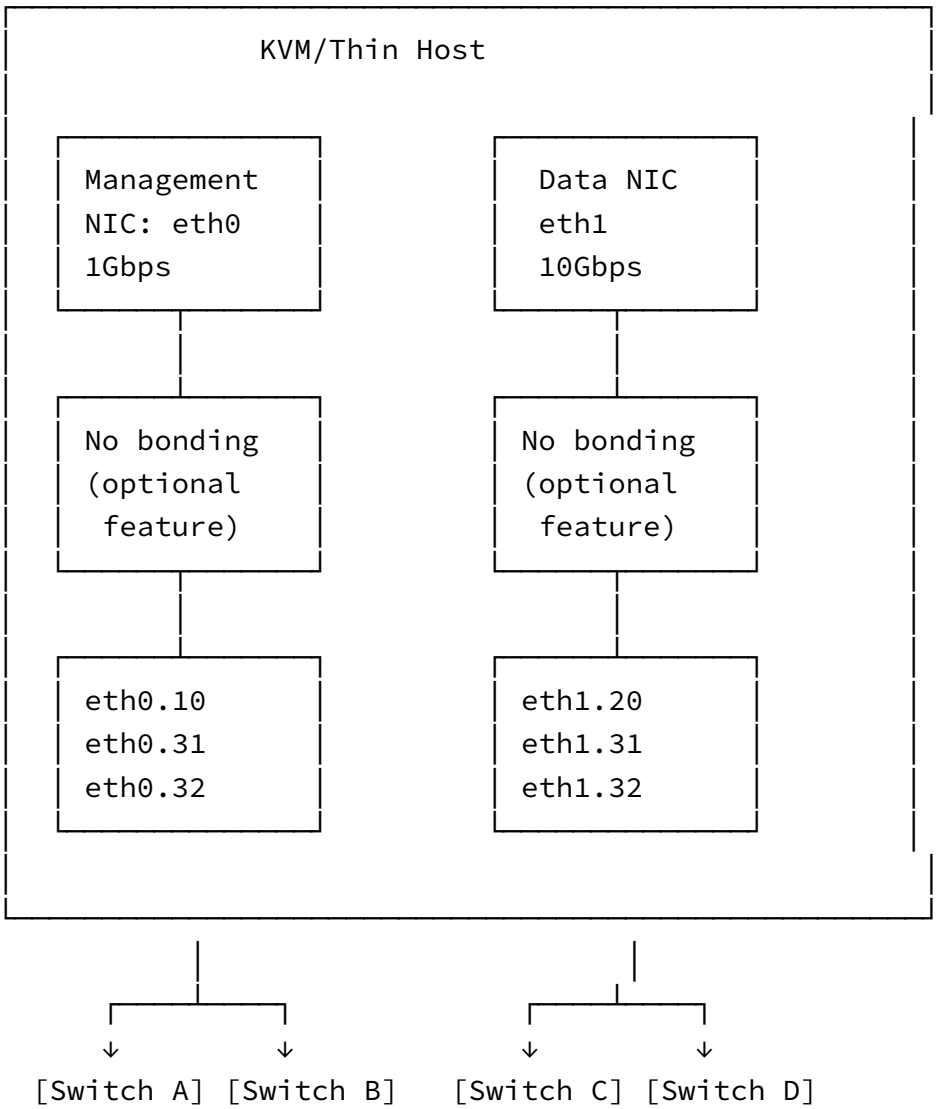
1. PXE boot from eth0 (untagged) → iPXE
2. iPXE creates eth0.10 (VLAN 10), gets IP
3. Downloads OS from VLAN 10
4. OS configures:
    - eth0.10 (management)
    - eth1.20 (VXLAN transport)
    - eth1.31, eth1.32 (storage)

9.4.3 Stage 3: Dual NIC, Dual Switch (High Availability)

Hardware:

- 1 x 1G NIC (management)
- 1 x 10G NIC (VXLAN + storage)
- 2 independent switches

Purpose: Production with switch-level redundancy



Optional Bonding Configuration:

Mode active-backup (no LACP required)
bond0 eth0 connects to Switch A and B (management failover)
bond1 eth1 connects to Switch C and D (data failover)
Failover Automatic upon link or switch failure

Characteristics:

- Switch-level redundancy (no single point of failure)
- Automatic failover if bonding used (typically <1 second)

- One active path per bond when bonding used (1G mgmt, 10G data)
- No switch coordination required (independent switches)
- Bonding optional - can run without for simpler configuration

Switch Requirements:

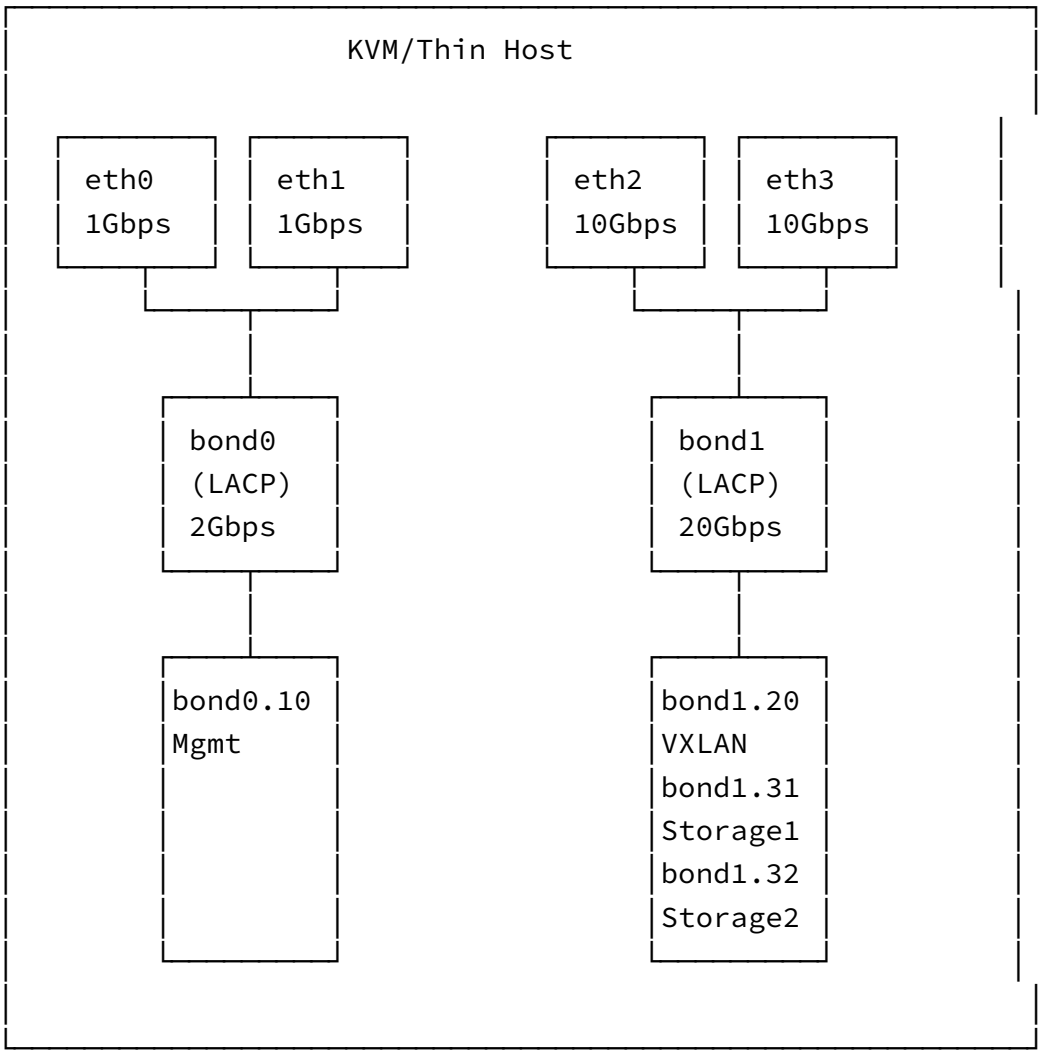
- Same as Stage 2
- Switches can be independent (no stacking/MC-LAG needed)
- If bonding used: same trunk configuration on both switches

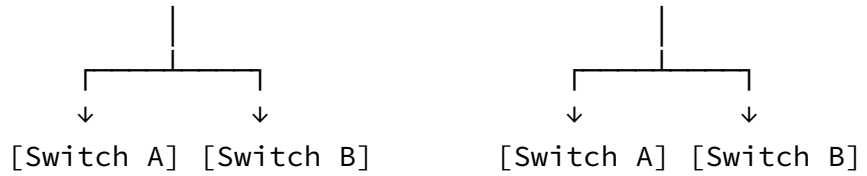
9.4.4 Stage 4: Quad NIC with LACP Bonding (Maximum Performance)

Hardware:

- 2 x 1G NICs (management bonding)
- 2 x 10G NICs (data bonding)
- 2 switches (can be independent)

Purpose: Production with bandwidth aggregation and redundancy





Note: Can use same switches or separate switch pairs for each bond

Bonding Configuration:

Mode 802.3ad (LACP)

bond0 eth0 + eth1 = 2Gbps aggregate (management). eth0 connects to Switch A, eth1 connects to Switch B.

bond1 eth2 + eth3 = 20Gbps aggregate (VXLAN + storage). eth2 connects to Switch A (or Switch C), eth3 connects to Switch B (or Switch D).

Hash policy layer3+4 (IP + port based distribution)

Characteristics:

- Maximum bandwidth: 22Gbps total (2G management + 20G data)
- Link-level redundancy within each bond
- Switch-level redundancy (each bond spans two switches)
- Load balancing across links (per-flow)
- Works with independent switches using standard LACP

Switch Requirements:

- LACP/802.3ad support (standard feature on managed switches)
- Each switch independently configured with LACP
- No switch stacking or MC-LAG required
- Traffic distributed per-flow between switches

Important: Each bond operates independently with standard LACP on each switch. This is the base Stage 4 configuration and works with any LACP-capable managed switches.

9.4.5 Optional Enhancement: MC-LAG for Stage 2+

MC-LAG (Multi-Chassis Link Aggregation) is an **optional enhancement** available for Stage 2, 3, or 4 when using advanced switches that support this feature.

MC-LAG allows two independent switches to coordinate and appear as a single logical switch for LACP purposes. This provides:

Benefits over standard LACP:

- Simpler host configuration (single bond instead of multiple independent bonds)
- Faster failover (sub-second vs 1-2 seconds)
- Active-active links (both links in bond actively used)
- No reconfiguration needed on switch failure

Requirements:

- Enterprise switches with MC-LAG capability
- Peer link between switches (high bandwidth)
- Vendor-specific configuration

Tradeoffs:

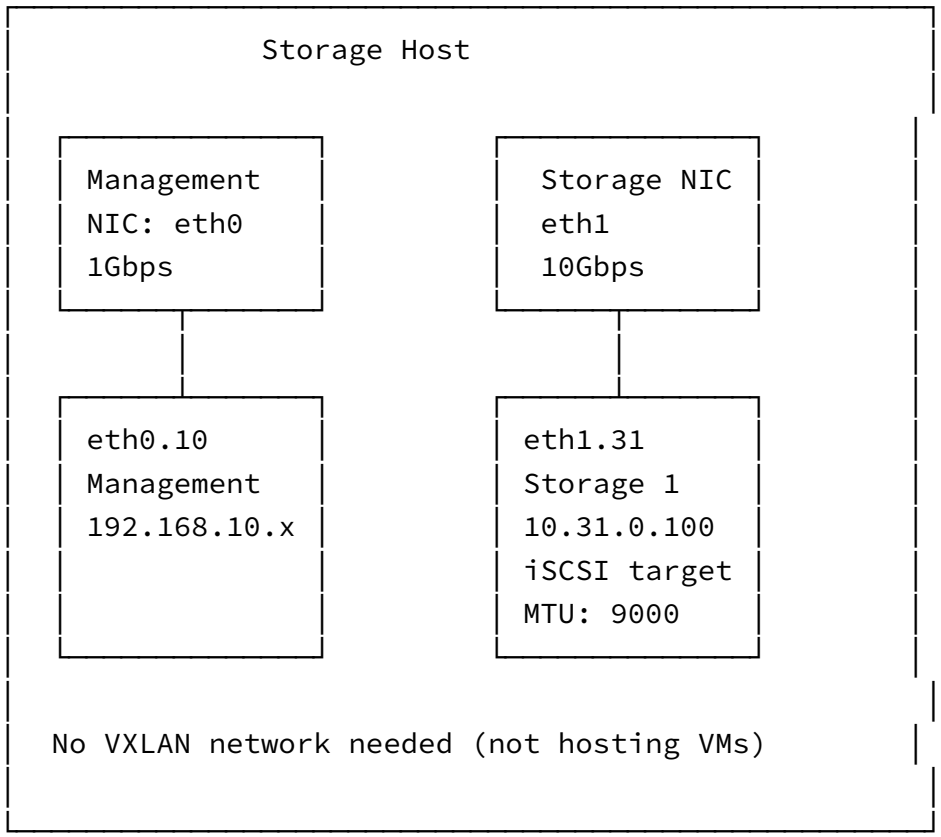
- More complex switch configuration
- Higher switch cost (enterprise feature)
- Vendor-specific setup

Recommendation: Start without MC-LAG using standard LACP (Stage 4 base configuration). This works with any LACP-capable managed switch and provides excellent redundancy and performance. Add MC-LAG later if you upgrade to enterprise switches with this capability and need sub-second failover for critical workloads.

9.5 Storage Host Considerations

9.5.1 Storage Host Network Configuration

Storage hosts (providing iSCSI targets) have similar but simplified networking:



VLANs used:

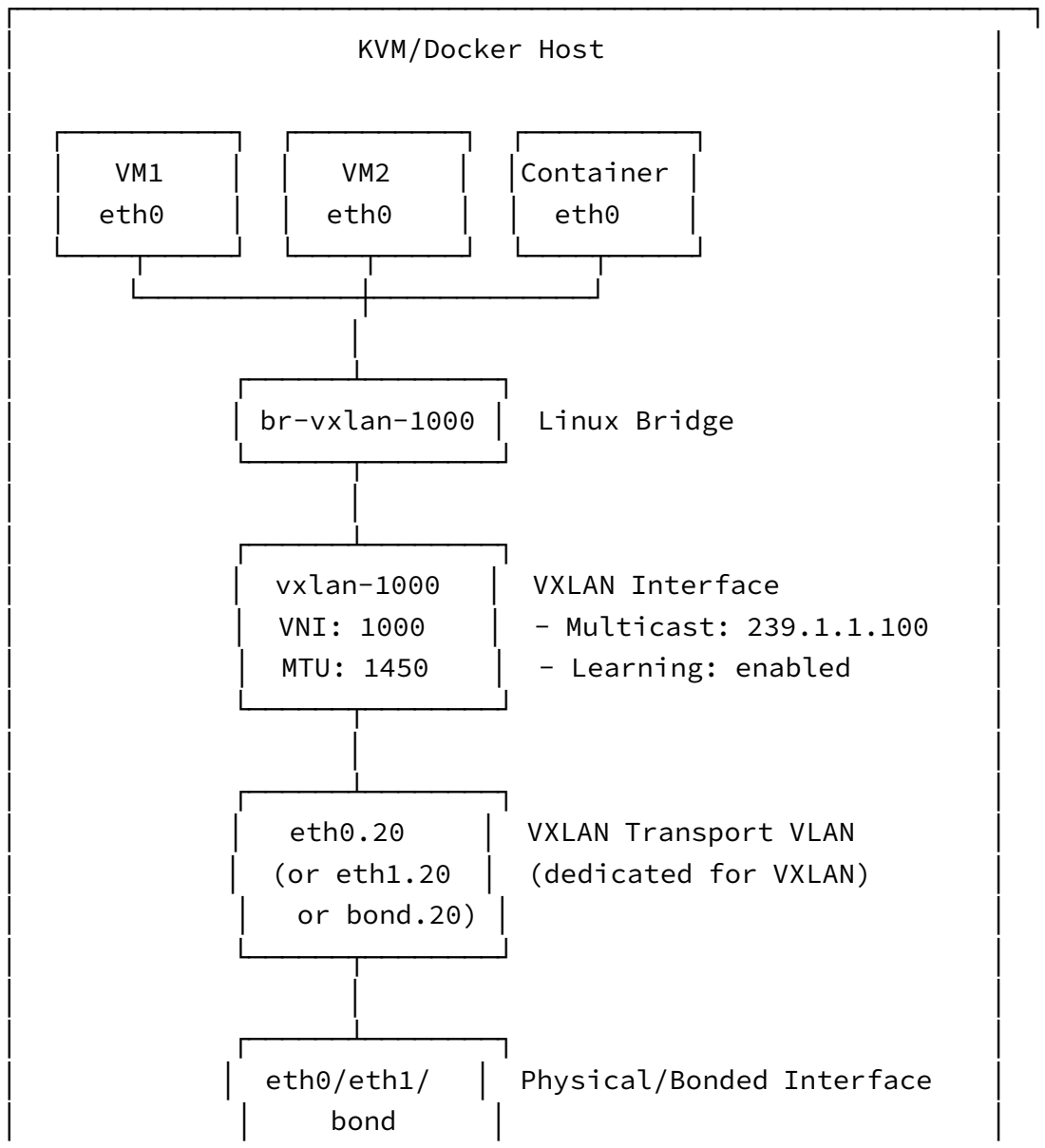
- VLAN 10: Management (SSH, monitoring)
- VLAN 31: Storage network for this host's iSCSI target
- No VLAN 20 (VXLAN) - storage hosts don't participate in customer networks

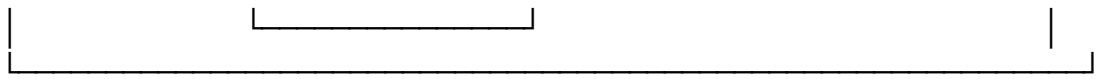
Each storage host uses one storage VLAN:

- Storage Host 1: VLAN 31 (10.31.0.100)
- Storage Host 2: VLAN 32 (10.32.0.100)
- Storage Host 3: VLAN 33 (10.33.0.100)
- etc.

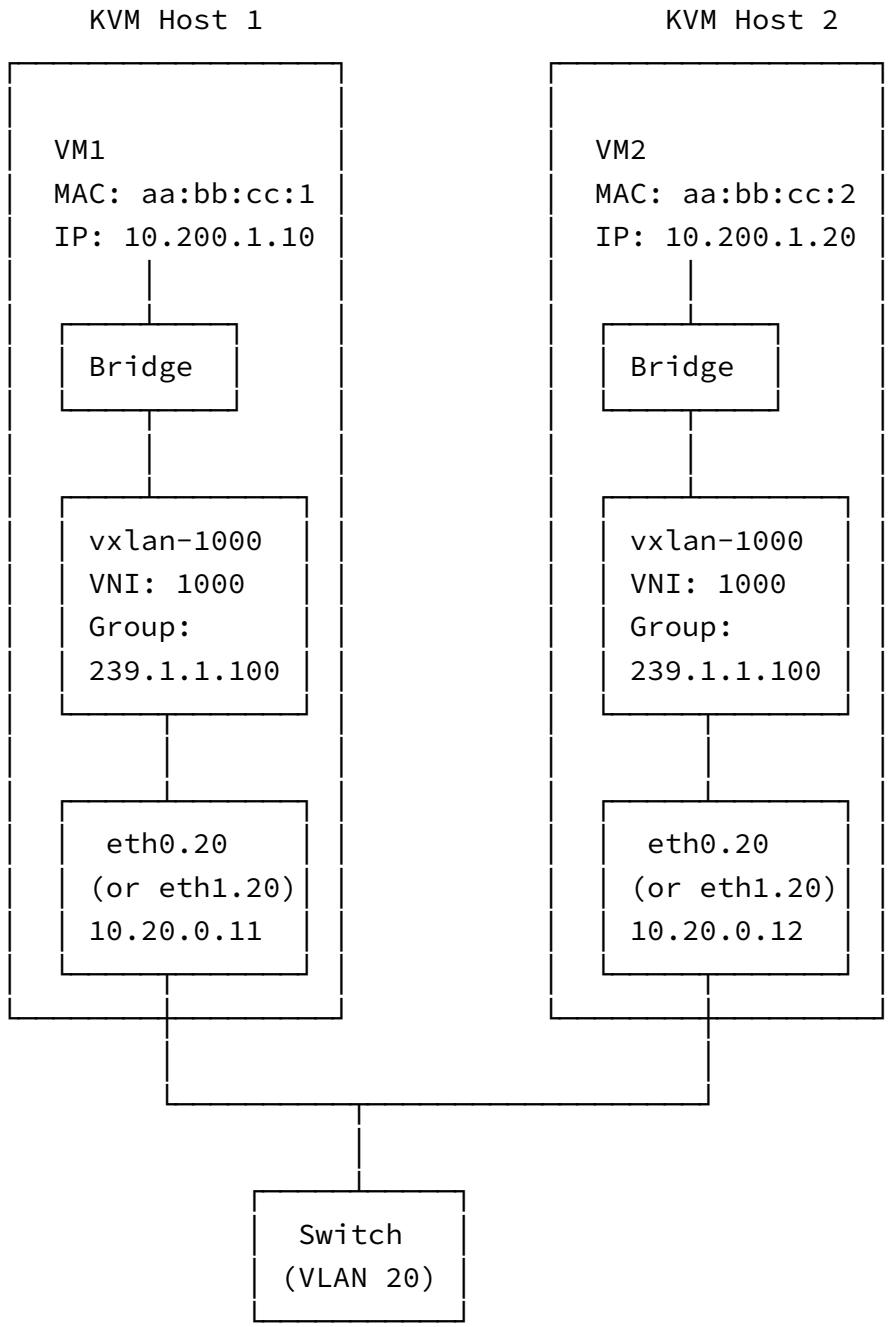
9.6 VXLAN Customer Network Architecture

9.6.1 Host-Level VXLAN Configuration





9.6.2 Multi-Host VXLAN Communication



Communication Flow:

1. VM1 sends to VM2 (10.200.1.20)
2. ARP: Multicast 239.1.1.100 → all VTEPs receive
3. VM2 responds, Host 1 learns MAC aa:bb:cc:2 at 10.20.0.12
4. Future frames: unicast encapsulation to 10.20.0.12
5. VXLAN overhead: 50 bytes (hence MTU 1450 for VMs)

9.6.3 VXLAN Packet Encapsulation

Inside VM (payload):

```
┌─────────────────────────────────────────────┐
│ Ethernet Header (14 bytes)                  │
│  Dest MAC: aa:bb:cc:2                       │
│  Src MAC:  aa:bb:cc:1                       │
├─────────────────────────────────────────────┤
│ IP Header (20+ bytes)                       │
│  Dest IP: 10.200.1.20                       │
│  Src IP:  10.200.1.10                       │
├─────────────────────────────────────────────┤
│ TCP/UDP + Application Data                  │
└─────────────────────────────────────────────┘
```

After VXLAN encapsulation (on wire):

```
┌─────────────────────────────────────────────┐
│ Outer Ethernet (14 bytes)                   │
│  Dest MAC: Switch MAC                       │
│  Src MAC:  Host 1 eth0.20 MAC               │
├─────────────────────────────────────────────┤
│ Outer IP (20 bytes)                         │
│  Dest IP: 10.20.0.12 (Host 2 on VLAN 20)    │
│  Src IP:  10.20.0.11 (Host 1 on VLAN 20)    │
├─────────────────────────────────────────────┤
│ Outer UDP (8 bytes)                         │
│  Dest Port: 4789 (VXLAN standard)           │
│  Src Port:  Random (flow hash)              │
├─────────────────────────────────────────────┤
│ VXLAN Header (8 bytes)                      │
│  VNI: 1000                                  │
├─────────────────────────────────────────────┤
│ Inner Ethernet + IP + Data (original frame) │
│  (Same as payload above)                    │
└─────────────────────────────────────────────┘
```

```
Total overhead: 50 bytes
VM MTU: 1500 - 50 = 1450 bytes
VXLAN transport VLAN 20 MTU: 1504 bytes (accommodates overhead)
```

Key Point: VXLAN traffic is isolated on VLAN 20, separate from management (VLAN 10) and storage (VLAN 31, 32). This provides clear traffic separation and allows independent QoS policies.

9.7 MTU Configuration Strategy

9.7.1 Per-Network MTU Settings

```
Network Layer                          MTU Setting
═══════════════════════════════════════════════════

Physical Interfaces (Stage 1):
├─ eth0                                1504 bytes

Physical Interfaces (Stage 2+):
├─ eth0 (management)                   1500 bytes
└─ eth1 (VXLAN + storage)              9000 bytes

Bonded Interfaces (Stage 3+):
├─ bond0 (management)                  1500 bytes
└─ bond1 (VXLAN + storage)             9000 bytes

VLAN Interfaces:
├─ vlan.10 (management)                1500 bytes
├─ vlan.20 (VXLAN transport)           1504 bytes
├─ vlan.31 (storage1)                  9000 bytes
├─ vlan.32 (storage2)                  9000 bytes
└─ vlan.33+ (additional storage)       9000 bytes

VXLAN Interfaces:
└─ vxlan-* (all customer VXLANs)       1450 bytes

Bridges:
├─ br-vxlan-* (customer bridges)       1450 bytes
└─ br-storage (if used)                9000 bytes

VM/Container Interfaces:
├─ Customer network VMs                1450 bytes
└─ Storage network VMs                 9000 bytes
```

9.7.2 Rationale

VLAN 20 (VXLAN transport): 1504 Accommodates VXLAN overhead (50 bytes) while keeping customer VM MTU at 1450

VLAN 10 (Management): 1500 Standard MTU, no special requirements

Storage VLANs (31, 32+): 9000 Jumbo frames improve iSCSI performance (10-15% throughput gain, reduced CPU)

VXLAN interfaces: 1450 Leaves headroom for encapsulation without fragmentation

Customer VMs: 1450 Standard for internet connectivity, prevents fragmentation issues

---

9.8 Switch Configuration Requirements

9.8.1 Minimum Requirements (All Stages)

VLAN Support:

- Trunk port configuration
- Multiple VLANs per port (10, 20, 31, 32 minimum)
- Native VLAN handling (recommend unused VLAN for security)

Frame Size:

- Support for 1522+ byte frames (VLAN tagged)
- Support for 1608+ byte frames (VXLAN on VLAN 20)
- Jumbo frame support on storage ports (9000+ bytes)

9.8.2 Stage 2+ Requirements

IGMP Snooping (recommended for VXLAN efficiency on VLAN 20):

Purpose Intelligent multicast forwarding
Without IGMP snooping Multicast flooded to all ports (works but inefficient)
With IGMP snooping Multicast only to interested ports

Configuration concept (vendor agnostic):

- Enable IGMP snooping globally
- Enable on VLAN 20 (VXLAN transport network)
- Optionally enable IGMP querier if no router present

Port Configuration (management network example):

```
Interface: Port connecting to host management NIC (eth0)
Mode: Trunk
Allowed VLANs: 10
Native VLAN: None (or unused VLAN 999)
Allow untagged: Yes (for bootstrap only, can be removed after)
MTU: 1600
```

Port Configuration (data network example):

```
Interface: Port connecting to host data NIC (eth1)
Mode: Trunk
Allowed VLANs: 20, 31, 32
MTU: 9216 (accommodates jumbo frames + overhead)
IGMP Snooping: Enabled (for VLAN 20)
Spanning Tree: Edge port (PortFast equivalent)
```

Port Configuration (storage host example):

```
Interface: Port connecting to storage host
Mode: Trunk
Allowed VLANs: 10, 31 (or 32, 33, etc.)
MTU: 9216
```

9.8.3 Stage 4 Base Requirements (Standard LACP)

Link Aggregation (LACP/802.3ad):

Purpose Bonding multiple links for bandwidth aggregation
Requirement Standard LACP support on managed switches

Configuration notes:

- Each switch independently configured
- No coordination between switches required
- Standard feature on managed switches
- Each bond member connects to different switch

LAG Configuration Concept (per switch):

```
Port-Channel/LAG: lag1
Members: port1 (from this switch only)
Mode: LACP Active
Load Balance: src-dst-ip-port
Allowed VLANs: 10, 20, 31, 32
MTU: 9216

Note: Each switch has its own LAG config
Host bond spans both switches independently
```

9.8.4 Vendor-Agnostic Configuration Checklist

For all switches supporting HPS:

- ☐ VLAN support with trunk ports
- ☐ MTU 1522+ on management ports
- ☐ MTU 1608+ on VXLAN transport ports (VLAN 20)
- ☐ MTU 9216+ on storage network ports (VLAN 31+)
- ☐ IGMP snooping available and enabled on VLAN 20
- ☐ Spanning tree configured (edge ports for host connections)
- ☐ Allow untagged traffic for bootstrap (can be removed after deployment)

For Stage 4 base deployments:

- ☐ LACP/802.3ad support (standard on managed switches)
- ☐ Per-switch LAG configuration

- ☐ No coordination between switches needed

For Stage 4 with MC-LAG enhancement (optional):

- ☐ Switch stacking OR MC-LAG capability
- ☐ Peer link configuration between switches
- ☐ Coordinated LAG across switch pair

9.9 Essential Configuration Commands

9.9.1 Bootstrap: iPXE VLAN Transition Script

Served to host during initial PXE boot:

```
#!ipxe
# HPS Bootstrap Script - Transition to VLAN 10

echo ========================================
echo HPS Network Bootstrap
echo ========================================

echo Configuring management VLAN 10
vcreate --tag 10 net0 || goto failed

echo Requesting IP address on VLAN 10
dhcp net0-10 || goto failed

echo Management network configured
echo IP: ${net0-10/ip}
echo Gateway: ${net0-10/gateway}

echo Chainloading main boot manager
chain http://${next-server}/cgi-bin/boot_manager.sh?phase=main ||
 ↪  goto failed

:failed
echo Bootstrap failed - dropping to iPXE shell
shell
```

9.9.2 Host-Level VXLAN Setup

Create VXLAN interface with multicast on VLAN 20:

```
ip link add vxlan-cust-a type vxlan \
  id 1000 \
  dstport 4789 \
  group 239.1.1.100 \
  dev eth0.20 \
```

```
  ttl 5 \
  learning

# Note: dev eth0.20 (or eth1.20) - VXLAN on dedicated VLAN 20
```

Create bridge and attach VXLAN:

```
ip link add br-vxlan-cust-a type bridge
ip link set br-vxlan-cust-a type bridge stp_state 0
ip link set vxlan-cust-a master br-vxlan-cust-a
```

Set MTU and bring up:

```
ip link set vxlan-cust-a mtu 1450
ip link set br-vxlan-cust-a mtu 1450
ip link set vxlan-cust-a up
ip link set br-vxlan-cust-a up
```

Verify multicast group membership:

```
ip maddr show dev eth0.20 | grep 239.1.1.100
```

Check learned MAC addresses:

```
bridge fdb show dev vxlan-cust-a
```

9.9.3 Bonding Configuration

Active-backup bond (Stage 3):

```
ip link add bond0 type bond mode active-backup
ip link set bond0 type bond miimon 100 primary eth0
ip link set eth0 master bond0
ip link set eth1 master bond0
ip link set bond0 up
```

LACP bond (Stage 4 base):

```
ip link add bond0 type bond mode 802.3ad
ip link set bond0 type bond miimon 100 lacp_rate fast
 ↪  xmit_hash_policy layer3+4
ip link set eth0 master bond0
ip link set eth1 master bond0
ip link set bond0 up
```

9.10 Network Component Summary

The following components require configuration management functions. Each represents a distinct functional area for automation:

9.10.1 1. Bootstrap Manager

Purpose: Handle diskless PXE boot and rapid VLAN transition

Responsibilities:

- Serve DHCP responses for untagged bootstrap requests
- Provide iPXE bootloader binary
- Generate iPXE VLAN configuration script
- Detect bootstrap vs runtime phase
- Serve appropriate boot files per phase
- Log bootstrap attempts and transitions

Key considerations:

- Untagged phase exposure (~3-7 seconds)
- VLAN 10 transition via iPXE
- Security mode handling (exploratory/production/high)
- Fallback mechanisms if iPXE fails

9.10.2 2. Physical Interface Management

Purpose: Detect, configure, and manage physical NICs (eth0, eth1, etc.)

Responsibilities:

- Enumerate available physical interfaces
- Detect link speed and status (1G vs 10G)
- Set MTU on physical interfaces based on purpose
- Configure promiscuous mode if needed
- Handle interface state (up/down)
- Identify management vs data interfaces

Key considerations:

- Stage detection (how many NICs available?)
- 1G management, 10G data differentiation
- Validation of physical connectivity
- MTU requirements per stage and purpose

9.10.3 3. VLAN Interface Management

Purpose: Create and manage 802.1Q VLAN interfaces on physical or bonded interfaces

Responsibilities:

- Create VLAN interfaces (vlan.10, vlan.20, vlan.31, vlan.32+)

- Map VLANs to physical interfaces based on deployment stage
- Set MTU on VLAN interfaces (1500 mgmt, 1504 VXLAN, 9000 storage)
- Handle VLAN interface lifecycle
- Validate VLAN tag configuration

Key considerations:

- Stage-specific VLAN-to-interface mapping
- Infrastructure VLANs (10, 20, 31, 32+) are predefined
- VLAN 10: Management
- VLAN 20: VXLAN transport (dedicated)
- VLAN 31+: Storage networks (one per storage host)
- MTU inheritance and override per VLAN purpose

9.10.4 4. Bonding/Link Aggregation Management

Purpose: Create and manage bonded interfaces for redundancy and bandwidth aggregation

Responsibilities:

- Create bond interfaces (bond0, bond1)
- Configure bonding mode (active-backup, 802.3ad)
- Add/remove slave interfaces
- Set bonding parameters (miimon, lacp_rate, xmit_hash_policy)
- Monitor bond status and failover
- Handle primary interface selection

Key considerations:

- Stage 3 uses active-backup (no switch dependency)
- Stage 4 uses 802.3ad/LACP (standard managed switch feature)
- Separate bonds for management and data
- Failover detection and recovery
- MC-LAG is optional enhancement (not required)

9.10.5 5. Bridge Management

Purpose: Create and manage Linux bridges for VM/container attachment

Responsibilities:

- Create bridge interfaces
- Configure bridge parameters (STP state, filtering)
- Attach interfaces to bridges (VLAN, VXLAN)
- Set MTU on bridges

- Manage bridge FDB (forwarding database)

Key considerations:

- Bridges for VXLAN customer networks
- Optional bridges for storage networks
- STP disabled for VXLAN bridges (no loops with VXLAN)
- MTU matching underlying interfaces

9.10.6 6. VXLAN Interface Management

Purpose: Create and manage VXLAN tunnel endpoints (VTEPs)

Responsibilities:

- Create VXLAN interfaces with VNI assignment
- Configure multicast groups
- Set VXLAN parameters (dstport, ttl, learning)
- Attach VXLAN to VLAN 20 transport interface
- Set MTU (1450) on VXLAN interfaces
- Monitor VXLAN FDB and remote VTEPs

Key considerations:

- VNI allocation per customer network
- Multicast group assignment (one per VNI)
- Transport over VLAN 20 (dedicated VXLAN transport VLAN)
- VLAN 20 on appropriate physical interface (eth0.20, eth1.20, or bond.20)
- MAC learning enabled for automatic peer discovery
- Isolation from management and storage traffic

9.10.7 7. IP Address Management

Purpose: Assign and manage IP addresses on interfaces

Responsibilities:

- Assign static IPs to infrastructure interfaces
- Configure subnet masks and network parameters
- Set default routes if needed
- Handle IP address conflicts
- Manage IP address pools for allocation
- Track bootstrap vs runtime IP assignments

Key considerations:

- Bootstrap network: 192.168.100.0/24 (untagged, temporary)

- Management network: 192.168.10.0/24 (VLAN 10)
- VXLAN transport: 10.20.0.0/24 (VLAN 20)
- Storage networks: 10.31.0.0/24, 10.32.0.0/24, etc. (VLAN 31+)
- Customer networks: 10.200.0.0/16+ (VXLAN overlays)
- Per-host IP assignments from registry

9.10.8 8. Routing and Policy Configuration

Purpose: Configure routing tables and policy routing

Responsibilities:

- Set default routes (if needed)
- Configure source-based routing (if multiple networks)
- Ensure storage traffic uses correct interface
- Ensure VXLAN traffic uses VLAN 20
- Prevent unwanted cross-network routing
- Configure IP forwarding state

Key considerations:

- Typically no routing needed (all services local)
- Disable IP forwarding unless host is router/firewall
- Firewall rules to prevent customer-to-infrastructure access
- Separate management, VXLAN, and storage paths

9.10.9 9. Multicast Configuration

Purpose: Enable and configure multicast support for VXLAN

Responsibilities:

- Enable multicast reception on VLAN 20 interfaces
- Configure IGMP parameters
- Join multicast groups (automatic via VXLAN, but validate)
- Monitor multicast group membership
- Handle multicast routing settings if needed

Key considerations:

- Required for VXLAN multicast mode on VLAN 20
- IGMP version compatibility
- Multicast TTL settings
- IGMP snooping on switches (VLAN 20)

9.10.10 10. MTU Management

Purpose: Configure Maximum Transmission Unit across network stack

Responsibilities:

- Set MTU on physical interfaces based on purpose
- Propagate MTU to VLAN interfaces
- Set appropriate MTU on VXLAN interfaces (1450)
- Configure MTU on VLAN 20 (1504 for VXLAN overhead)
- Configure MTU on storage VLANs (9000 for jumbo frames)
- Validate MTU path between hosts

Key considerations:

- Stage 1: 1504 on eth0 (accommodates all traffic)
- Stage 2+: 1500 on management NIC, 9000 on data NIC
- VLAN 10 (management): 1500
- VLAN 20 (VXLAN transport): 1504
- VLAN 31+ (storage): 9000
- VXLAN interfaces: 1450 (accounting for 50-byte overhead)
- Path MTU discovery testing

9.10.11 11. Network Validation and Testing

Purpose: Verify network configuration correctness and connectivity

Responsibilities:

- Validate interface state (up/down, link speed)
- Test VLAN connectivity between hosts
- Verify VXLAN tunnel establishment on VLAN 20
- Check multicast group membership
- Test MTU end-to-end (ping with DF flag)
- Validate bridge FDB learning
- Monitor for errors/drops
- Verify bootstrap to runtime transition

Key considerations:

- Pre-deployment validation
- Bootstrap phase verification
- VLAN 10 management connectivity
- VLAN 20 VXLAN multicast functionality
- Storage VLAN (31+) jumbo frame support
- Continuous health monitoring
- Troubleshooting support

- Performance metrics collection

9.10.12 12. Network Registry/Configuration Management

Purpose: Store and manage network definitions and host assignments

Responsibilities:

- Define logical networks (management, VXLAN transport, storage)
- Map networks to VLANs (10, 20, 31+)
- Assign VNI numbers to customer networks
- Store per-host interface assignments
- Track IP address allocations
- Store multicast group assignments
- Maintain deployment stage configuration
- Track bootstrap security mode

Key considerations:

- Central source of truth for network config
- VLAN numbering scheme (10, 20, 31+)
- Storage VLAN allocation (one per storage host)
- VXLAN on dedicated VLAN 20
- Version control of network definitions
- Cluster-specific configurations
- Customer network isolation mappings
- Bootstrap mode configuration

9.10.13 13. VM/Container Network Attachment

Purpose: Attach VMs and containers to customer network bridges

Responsibilities:

- Configure libvirt/KVM network definitions
- Attach VM interfaces to specific bridges
- Create veth pairs for container attachment
- Set MAC addresses
- Configure MTU on VM/container interfaces (1450 for VXLAN networks)
- Handle hot-plug/unplug if needed

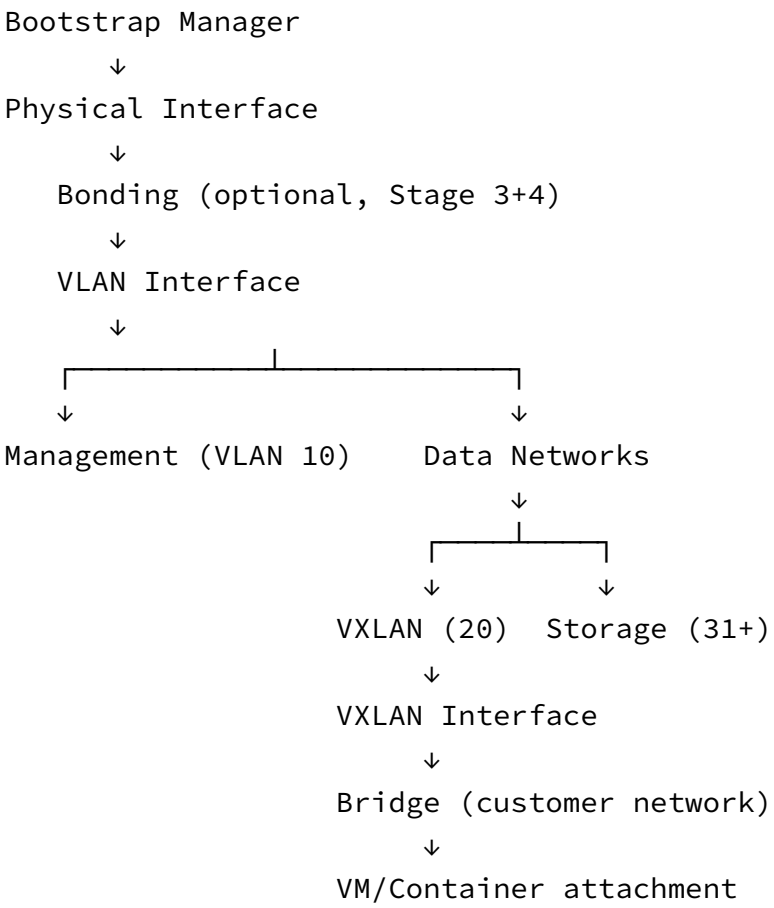
Key considerations:

- Libvirt XML generation
- Docker network integration
- Customer network isolation enforcement

- MAC address management
- MTU settings (1450 for customer VMs on VXLAN)

9.11 Integration Points

These components interact as follows:



Each component should be implemented as independent, testable functions that can be composed to build the complete network stack based on the deployment stage.

9.12 Deployment Stage Decision Matrix

Available Hardware	Recommended Stage	Management NIC
1 NIC per host Any switch	Stage 1 (Testing only) No switch config	Shared 10G Bootstrap: untagged Runtime: VLAN 10
1G + 10G NIC per host	Stage 2 (Production)	Dedicated 1G

```
1 managed switch          Configure VLANs      Bootstrap: untagged
                          IGMP snooping         Runtime: VLAN 10

1G + 10G NIC per host        Stage 3 (HA)           Dedicated 1G
2 independent switches     Configure VLANs       Optional bonding
                        Optional bonding      Bootstrap: untagged

2x1G + 2x10G per host        Stage 4 (Max perf)     Bonded 2x1G
2 switches with LACP       LACP bonding         Bootstrap: untagged
                          VLANs + LAG            Runtime: bond.10
```

9.12.1 VLAN Usage Summary by Stage

Stage 1 (Single 10G NIC):

- VLAN 10: Management (eth0.10)
- VLAN 20: VXLAN transport (eth0.20)
- VLAN 31: Storage 1 (eth0.31)
- VLAN 32: Storage 2 (eth0.32)

Stage 2 (1G mgmt + 10G data):

- VLAN 10: Management (eth0.10) - on 1G NIC
- VLAN 20: VXLAN transport (eth1.20) - on 10G NIC
- VLAN 31: Storage 1 (eth1.31) - on 10G NIC
- VLAN 32: Storage 2 (eth1.32) - on 10G NIC

Stage 3 (with optional bonding):

- Same as Stage 2, but interfaces may be bonded
- VLAN 10: Management (bond0.10)
- VLAN 20: VXLAN transport (bond1.20)
- VLAN 31: Storage 1 (bond1.31)
- VLAN 32: Storage 2 (bond1.32)

Stage 4 (full bonding):

- VLAN 10: Management (bond0.10) - on 2x1G bonded
- VLAN 20: VXLAN transport (bond1.20) - on 2x10G bonded
- VLAN 31: Storage 1 (bond1.31) - on 2x10G bonded
- VLAN 32: Storage 2 (bond1.32) - on 2x10G bonded

Each stage builds upon the same logical network definitions (VLANs 10, 20, 31, 32+), differing only in physical interface mapping and redundancy features. This allows progressive hardware enhancement without reconfiguring the logical network layer.# KV Database Library Specification

File: `lib/functions.d/kvdb-functions.sh`

9.13 Overview

Treat config files as a Key-Value database with proper locking, scanning, and atomic operations. Separate from general file management tools but shares locking mechanisms.

9.14 Core Configuration

9.14.1 Environment Variables

```
KVDB_LOCK_STALE_TIMEOUT=${KVDB_LOCK_STALE_TIMEOUT:-5}
KVDB_LOCK_ACQUIRE_TIMEOUT=${KVDB_LOCK_ACQUIRE_TIMEOUT:-30}
KVDB_LOCK_RETRY_INTERVAL=${KVDB_LOCK_RETRY_INTERVAL:-1}
KVDB_LOCK_RETRY_JITTER=${KVDB_LOCK_RETRY_JITTER:-500}  #
 ↪  milliseconds
```

9.14.2 Lock File Format

`PID:EPOCH:HOSTNAME:FUNCTION`

Example: `12345:1696248372:docker-01:kvdb_set`

9.14.3 Lock File Locations

```
/path/to/file.conf          → /path/to/.file.conf.lock
/path/to/directory/         → /path/to/directory/.dir.lock
```

9.15 Lock Management Functions

9.15.1 kvdb_acquire_lock(file_path, lock_type, stale_timeout, acquire_timeout)

Purpose: Acquire file or directory lock with retry and jitter

Arguments: - $1 file_path - Path to file or directory - $2 lock_type - “file” | “dir” | “both” (default: “file”) - $3 stale_timeout - Seconds before lock considered stale (default: 5) - $4 acquire_timeout - Seconds to retry before giving up (default: 30)

Behavior: 1. Determine lock file path(s) based on lock_type 2. Loop with timeout: - Check for existing lock file - If exists: - Parse PID:EPOCH:HOSTNAME:FUNCTION - Check if process still alive: `kill -0 $PID 2>/dev/null` - Check age: `current_epoch - lock_epoch > stale_timeout` - If stale (dead process OR too old): remove lock - If valid: sleep 1s + random jitter (0-500ms), retry - If not exists: create lock file - Verify lock

creation succeeded and contains our PID - Return success 3. After acquire_timeout: fail with error

Lock File Content:

```
echo "$$:$(date +%s):$(hostname):${FUNCNAME[2]}" > "$lockfile"
```

Returns: - 0 on success - 1 on timeout - 2 on invalid arguments

Logging: - debug: Each retry attempt - warn: Removed stale lock (show previous owner details) - error: Acquisition timeout

Notes: - Not safe for NFS or network filesystems (add comment warning) - Random jitter prevents thundering herd on lock contention

---

9.15.2 kvdb_release_lock(file_path, lock_type)

Purpose: Release previously acquired lock

Arguments: - $1 file_path - Path to file or directory - $2 lock_type - “file” | “dir” | “both” (default: “file”)

Behavior: 1. Determine lock file path(s) 2. For each lock file: - Verify exists - Read lock content - Parse PID from lock - If PID matches $$: remove lock file - If PID doesn’t match: log warning (someone else’s lock) 3. Return success if all locks released

Returns: - 0 on success - 1 if lock not found or not owned by current process

Logging: - warn: Attempted to release lock not owned by us - debug: Lock released successfully

---

9.16 Basic KV Operations

9.16.1 kvdb_get(db_file, key)

Purpose: Read value for a single key

Arguments: - $1 db_file - Path to config file - $2 key - Key name to retrieve

Behavior: 1. Validate db_file exists 2. Validate key format: `[A-Za-z_][A-Za-z0-9_]*` 3. Read file, find line matching ^KEY= 4. Extract value, strip quotes 5. Output value to stdout

Returns: - 0 if key found (outputs value) - 1 if key not found or file doesn’t exist

Locking: None (single atomic read)

Notes: - Must handle both quoted and unquoted values - Must handle escaped quotes in values

---

9.16.2 kvdb_set(db_file, key, value, lock_type, create_file)

Purpose: Set or update a key-value pair

Arguments: - $1 db_file - Path to config file - $2 key - Key name - $3 value - Value to set - $4 lock_type - “file” | “dir” | “both” | “none” (default: “file”) - $5 create_file - “true” | “false” (default: “true”)

Behavior: 1. Validate key format 2. Check directory exists (fail if not) 3. If file doesn’t exist and create_file=“false”: fail 4. If lock_type != “none”: Acquire lock 5. If file doesn’t exist: create with header 6. Read file into memory/temp 7. If key exists: replace line 8. If key doesn’t exist: append line 9. Ensure value properly quoted and escaped 10. Write temp file 11. Atomic move: `mv -f tempfile db_file` 12. If lock_type != “none”: Release lock

Returns: - 0 on success - 1 on lock failure - 2 on write failure - 3 on validation failure

Value Quoting: - Always quote values: KEY="value" - Escape embedded quotes: " → \" - Escape backslashes: \ → \\

9.16.3 kvdb_delete(db_file, key, lock_type)

Purpose: Remove a key from config file

Arguments: - $1 db_file - Path to config file - $2 key - Key to remove - $3 lock_type - “file” | “dir” | “both” | “none” (default: “file”)

Behavior: 1. Validate file exists 2. If lock_type != “none”: Acquire lock 3. Read file, filter out lines matching ^KEY= 4. Write to temp file 5. Atomic move 6. If lock_type != “none”: Release lock

Returns: - 0 on success (even if key didn’t exist) - 1 on lock failure - 2 on write failure

---

9.16.4 kvdb_exists(db_file, key)

Purpose: Check if key exists in file

Arguments: - $1 db_file - Path to config file - $2 key - Key name

Behavior: 1. Check file exists 2. grep for ^KEY= in file

Returns: - 0 if key exists - 1 if key doesn’t exist or file missing

Locking: None

9.17 Scanning Operations (Directory Lock Required)

9.17.1 kvdb_scan(db_pattern, key, value_pattern)

Purpose: Scan multiple files for matching key-value pairs

Arguments: - $1 db_pattern - Glob pattern (e.g., “hosts/*.conf”) - *$2 key - Key to search for (“”* for all keys) - $3 value_pattern - Pattern to match (“*” for any value)

Behavior: 1. Extract directory from pattern 2. Acquire directory lock 3. Expand glob pattern to file list 4. For each file: - If key=“*“: read all keys - Else: read specific key - If value_pattern matches: output”filename:key:value” 5. Release directory lock

Returns: - 0 on success (outputs matches to stdout, one per line) - 1 on lock failure

Output Format:

```
/path/to/file1.conf:IP:10.99.1.2
/path/to/file2.conf:IP:10.99.1.3
```

TODO: - Support regex patterns on keys and values - Add kvdb_list_keys(db_file) to enumerate all keys in one file - Add kvdb_dump(db_file) for debugging/export

---

9.17.2 kvdb_scan_values(db_pattern, key)

Purpose: Extract all values for a key across multiple files

Arguments: - $1 db_pattern - Glob pattern - $2 key - Key to extract values for

Behavior: 1. Extract directory from pattern 2. Acquire directory lock 3. Expand glob pattern 4. For each file: extract value for key 5. Output values (one per line) 6. Release directory lock

Returns: - 0 on success - 1 on lock failure

Example:

```
# Get all IPs
kvdb_scan_values "hosts/*.conf" "IP"
# Output:
10.99.1.2
10.99.1.3
10.99.1.4
```

9.17.3 kvdb_value_exists(db_pattern, key, value, exclude_file)

Purpose: Check if a value exists anywhere in matched files

Arguments: - $1 db_pattern - Glob pattern - $2 key - Key name - $3 value - Value to search for - $4 exclude_file - File to exclude from search (optional, for update scenarios)

Behavior: 1. Extract directory from pattern 2. Acquire directory lock 3. Expand glob pattern 4. For each file (except exclude_file): - Check if key=value exists - If found: release lock, return 0 5. Release directory lock 6. Return 1 (not found)

Returns: - 0 if value exists - 1 if value not found - 2 on lock failure

---

9.18 Unique Value Operations

9.18.1 kvdb_insert_if_unique(db_pattern, target_db, key, value, lock_type)

Purpose: Insert key-value only if value is unique across all files

Arguments: - $1 db_pattern - Pattern to scan for uniqueness (e.g., "hosts/*.conf") - $2 target_db - Specific file to write to (e.g., "hosts/12:34:56:78:9a:bc.conf") - $3 key - Key name (e.g., "IP") - $4 value - Proposed value (e.g., "10.99.1.5") - $5 lock_type - "file" | "dir" | "both" (default: "dir")

Behavior: 1. Extract directory from pattern 2. **Acquire directory lock** (critical for atomicity) 3. Call `kvdb_value_exists(db_pattern, key, value, target_db)` 4. If value exists in another file: - Release lock - Return failure 5. If value is unique: - Call `kvdb_set(target_db, key, value, "none", true)` - Note: "none" because dir lock already held - Release lock - Return success

Returns: - 0 on success (value was unique and inserted) - 1 on failure (value already exists elsewhere) - 2 on lock failure - 3 on write failure

Use Cases:

```
# Allocate unique IP
kvdb_insert_if_unique \
  "hosts/*.conf" \
  "hosts/${mac}.conf" \
  "IP" \
  "10.99.1.2" \
  "dir"

# Allocate unique hostname
kvdb_insert_if_unique \
  "hosts/*.conf" \
  "hosts/${mac}.conf" \
  "HOSTNAME" \
  "tch-001" \
  "dir"
```

Notes: - Caller must generate candidate value - Function only validates uniqueness and inserts - Directory lock ensures no race conditions

---

9.18.2 kvdb_get_next_sequence(db_pattern, key, prefix)

Purpose: Find next number in a sequence based on existing values

Arguments: - $1 db_pattern - Pattern to scan - $2 key - Key to examine (e.g., "HOSTNAME") - $3 prefix - Prefix to match (e.g., "tch-")

Behavior: 1. Acquire directory lock 2. Scan all files for key values 3. Filter values matching prefix 4. Extract numeric suffix from each 5. Find maximum number 6. Return max + 1 7. Release lock

Returns: - 0 on success (outputs next number to stdout) - 1 on lock failure

Example:

```
# Files contain: tch-001, tch-002, tch-004
next=$(kvdb_get_next_sequence "hosts/*.conf" "HOSTNAME" "tch-")
echo $next  # Outputs: 5
```

Notes: - Handles zero-padded numbers (001, 002, etc.) - Returns 1 if no existing values found - Does NOT insert the value (caller must use kvdb_insert_if_unique)

9.19 Index Management (Performance Optimization)

9.19.1 kvdb_index_build(db_pattern, key, index_file)

Purpose: Build an index of key values for fast lookups

Arguments: - $1 db_pattern - Pattern to index - $2 key - Key to index - $3 index_file - Path to index file (e.g., ".index.ip")

Behavior: 1. Acquire directory lock 2. Scan all files matching pattern 3. Extract key values 4. Build index: `value -> filename` mapping 5. Write to index file atomically 6. Release lock

Index File Format:

```
# Auto-generated index for key: IP
# Generated: 2025-10-02 13:00:00
10.99.1.2=/path/to/host1.conf
10.99.1.3=/path/to/host2.conf
```

Returns: - 0 on success - 1 on failure

TODO: Implement index management - kvdb_index_lookup(index_file, value) - Fast value lookup - kvdb_index_invalidate(index_file) - Mark index stale - kvdb_index_auto_update() - Rebuild if stale

---

9.19.2 `kvdb_register_indexed_key(key)`

Purpose: Register a key to be automatically indexed

Behavior: - Add key to list of indexed keys - Automatically rebuild indexes when files change

TODO: Implement automatic index management

9.20 Watch/Notification System

9.20.1 `kvdb_watch(db_pattern, callback_function)`

Purpose: Monitor files for changes and trigger callback

Arguments: - `$1` db_pattern - Pattern to watch - `$2` callback_function - Function to call on change

Behavior: - Use inotify or polling to detect changes - Call callback with changed file path

TODO: Spec this out after file writes complete

Example use case: Wait for new unconfigured host to boot and automatically configure it.

9.21 Helper Functions

9.21.1 `kvdb_validate_key(key)`

Purpose: Validate key name format

Validation Rules: - Must match: `^[A-Za-z_][A-Za-z0-9_]*$` - Must be bash variable compatible - Cannot be empty

Returns: - 0 if valid - 1 if invalid

9.21.2 `kvdb_escape_value(value)`

Purpose: Properly escape value for config file

Behavior: - Escape backslashes: \ → \\ - Escape quotes: " → \" - Return escaped value

---

9.21.3 `kvdb_unescape_value(value)`

Purpose: Unescape value read from config file

Behavior: - Remove surrounding quotes - Unescape \" → " - Unescape \\ → \

9.22 Error Codes

```
KVDB_SUCCESS=0
KVDB_LOCK_TIMEOUT=1
KVDB_WRITE_FAILURE=2
KVDB_VALIDATION_ERROR=3
KVDB_NOT_FOUND=4
KVDB_ALREADY_EXISTS=5
```

9.23 Usage Examples

9.23.1 Basic Operations

```
# Set a value
kvdb_set "host.conf" "IP" "10.99.1.2"

# Get a value
ip=$(kvdb_get "host.conf" "IP")

# Delete a key
kvdb_delete "host.conf" "OLD_KEY"

# Check if key exists
if kvdb_exists "host.conf" "HOSTNAME"; then
  echo "Hostname configured"
fi
```

9.23.2 Scanning

```
# Find all TCH hosts
kvdb_scan "hosts/*.conf" "TYPE" "TCH"
```

```
# Get all IPs
kvdb_scan_values "hosts/*.conf" "IP"

# Check if IP in use
if kvdb_value_exists "hosts/*.conf" "IP" "10.99.1.5"; then
  echo "IP already assigned"
fi
```

9.23.3 Unique Allocation

```
# Try to allocate IP
if kvdb_insert_if_unique "hosts/*.conf" "hosts/${mac}.conf" "IP"
 ↪  "10.99.1.5"; then
  echo "IP allocated successfully"
else
  echo "IP already in use"
fi

# Get next hostname number
next_num=$(kvdb_get_next_sequence "hosts/*.conf" "HOSTNAME"
 ↪  "tch-")
hostname="tch-$(printf '%03d' $next_num)"

# Try to allocate hostname
kvdb_insert_if_unique "hosts/*.conf" "hosts/${mac}.conf"
 ↪  "HOSTNAME" "$hostname"
```

9.24 Integration with Existing Functions

9.24.1 host_config refactor

```
host_config "$mac" set "IP" "10.99.1.2"
  ↓
kvdb_set "$(get_host_conf_filename $mac)" "IP" "10.99.1.2" "file"
 ↪  "true"
```

9.24.2 cluster_config refactor

```
cluster_config set "DHCP_IP" "10.99.1.1"
  ↓
kvdb_set "$(get_cluster_config_file)" "DHCP_IP" "10.99.1.1" "file"
 ↪  "true"
```

9.24.3 host_network_configure refactor

```
# Instead of get_cluster_host_ips + manual checking
kvdb_insert_if_unique "hosts/*.conf" "hosts/${mac}.conf" "IP"
 ↪  "$candidate_ip"

# Instead of get_cluster_host_hostnames + manual checking
kvdb_insert_if_unique "hosts/*.conf" "hosts/${mac}.conf"
 ↪  "HOSTNAME" "$candidate_hostname"
```

9.25 Testing Checklist

- ☐ Lock acquisition with retry and jitter
- ☐ Stale lock detection and removal
- ☐ Concurrent writes (race condition testing)
- ☐ Unique value validation across multiple files
- ☐ Directory vs file locking
- ☐ Proper quoting and escaping
- ☐ Missing file/directory handling
- ☐ Invalid key format rejection
- ☐ Lock cleanup on error
- ☐ Performance with many files

9.26 Implementation Notes

- Indent 2 spaces
- Follow HPS function documentation standards
- Use `hps_log` for all logging
- All functions in `lib/functions.d/kvdb-functions.sh`
- Test each function individually before integration
- Backup is not required (rely on write operation logs)
- NFS/network filesystems are NOT supported (document this)

9.27 Future Enhancements (TODO)

1. **Type validation** - Specify parameter types and re-validate (e.g., IP must be valid IP address)
2. **Query language** - Support complex queries like `TYPE=TCH AND STATE=CONFIGURED`

3. **Regex patterns** - Support regex on keys and values in scanning
4. **List/dump functions** - `kvdb_list_keys()`, `kvdb_dump()` for debugging
5. **Index management** - Automatic index building and maintenance
6. **Watch system** - File change notifications for auto-configuration# Working documents

This folder contains specifications and documents used during development phase.

These documents are almost certainly incomplete, erroneous and may be partly or not implemented.

However they are provided to show intent and insight in to the development process. # VM Provisioning Functions - Implementation Specification

9.28 Overview

This document specifies the implementation of VM provisioning functions for the HPS (High Performance System) infrastructure. These functions enable automated VM lifecycle management on Thin Compute Host (TCH) nodes using KVM/virsh through OpenSVC task orchestration.

Version: 2.1
Date: 2025-10-29
Status: Implemented and Tested

Changes from v2.0: - Updated to use official OpenSVC `status.gen` API for node reachability (confirmed by OpenSVC team) - Added comprehensive guide for choosing between JSON and flat output formats - Documented optimal parsing strategies with examples - Removed heartbeat API approach in favor of simpler status.gen method

Changes from v1.0: - Added node validation layer with health checks - Implemented using OpenSVC structured output formats (JSON and flat) - Added support for node reachability and frozen state detection - Implemented all node-side lifecycle functions (start, stop, pause, unpause, destroy) - Comprehensive test suite completed and passing - Updated exit codes to accommodate node validation failures

---

9.29 Architecture

9.29.1 Data Flow

```
IPS Function Call (o_vm_create)
         ↓
Validate Target Node Health
         ↓
Create Transient OpenSVC Service (nodes: ips + target)
```

```
         ↓
Add Task (n_vm_create <vm_id>)
         ↓
Wait for Service Instance on Target Node (30s timeout)
         ↓
Execute Task on Target TCH Node
         ↓
n_vm_create calls n_ips_command vm get_config
         ↓
Parse VM Configuration (KV pairs)
         ↓
Build virt-install Command
         ↓
Execute virsh Command
         ↓
Delete Transient Service
         ↓
Return Result
```

9.29.2 Component Interaction

IPS Layer (o_vm_* functions): - Validate target node health and availability - Orchestrate VM operations - Create/manage transient OpenSVC services - Handle error conditions and cleanup - Log all operations

Node Validation Layer (o_vm_validate_* functions): - Check node exists in cluster - Verify node daemon reachability via heartbeat generation counters - Check node frozen state - Filter healthy nodes from lists

Node Layer (n_vm_* functions): - Execute virsh commands - Fetch VM configuration from IPS - Report results back to IPS - Manage VM lifecycle (start, stop, pause, unpause, destroy)

IPS Command Interface: - Provides VM configuration data - Returns key=value text format - Supports flexible disk and network configs

---

9.30 OpenSVC API Usage - Best Practices

9.30.1 Overview of Output Formats

OpenSVC commands support multiple output formats for different use cases:

```
-o, --output string     output format json|flat|auto|tab=:,...
 ↪  (default "auto")
```

Available formats: - `json` - Structured JSON format (RECOMMENDED for complex data extraction) - `flat` - Key=value pairs (RECOMMENDED for simple single-value extraction) - `auto` - Human-readable tables (NOT for parsing - display only) - `tab=<sep>` - Tab-separated values with custom separator

9.30.2 When to Use Each Format

9.30.2.1 Use JSON Format When:

✅ Extracting nested/complex data structures

```
# Example: Get all heartbeat peer generation counters
om daemon status -o json | jq '.cluster.node.tch001.status.gen'
# Returns: {"ips": 7854, "tch-001": 9852, "tch-002": 9818}
```

✅ Working with arrays or objects

```
# Example: Get all node names in cluster
om daemon status -o json | jq -r '.cluster.node | keys[]'
```

✅ Need to process multiple related values together

```
# Example: Get node status bundle
om daemon status -o json | jq '.cluster.node.tch001.status | {gen,
 ↪  frozen_at, is_leader}'
```

✅ Performing calculations or transformations

```
# Example: Count nodes with generation data
om daemon status -o json | jq '[.cluster.node | to_entries[] |
 ↪  select(.value.status.gen)] | length'
```

9.30.2.2 Use Flat Format When:

✅ Extracting a single, specific value

```
# Example: Get daemon nodename
om daemon status --output flat | grep "^daemon\.nodename = " | cut
 ↪  -d'"' -f2
```

✅ Simple existence checks

```
# Example: Check if a key exists
if om daemon status --output flat | grep -q
 ↪  "^cluster\.node\.tch-001\.status\.gen\."; then
  echo "Node has gen data"
fi
```

✅ Counting specific patterns

```
# Example: Count gen entries for a node
om daemon status --output flat | grep -c
 ↪  "^cluster\.node\.tch-001\.status\.gen\."
```

✅ **Memory-constrained environments** - Flat output can be processed line-by-line without loading entire structure - Uses less memory than parsing large JSON documents

9.30.3 Parsing Best Practices

9.30.3.1 JSON Parsing with jq

Basic extraction:

```
# Extract single value
value=$(om daemon status -o json | jq -r
 ↪  '.cluster.node.ips.status.agent')

# Extract with fallback for missing keys (returns empty string,
 ↪  not "null")
value=$(om daemon status -o json | jq -r
 ↪  '.cluster.node.ips.status.agent // empty')

# Extract nested object
gen_data=$(om daemon status -o json | jq -r
 ↪  '.cluster.node.ips.status.gen')
```

Conditional filtering:

```
# Get all nodes with frozen_at not equal to zero value
om daemon status -o json | jq -r '
  .cluster.node |
  to_entries[] |
  select(.value.status.frozen_at != "0001-01-01T00:00:00Z") |
  .key
'
```

Array processing:

```
# Get all node names
nodes=$(om daemon status -o json | jq -r '.cluster.node | keys[]')

# Count nodes
node_count=$(om daemon status -o json | jq '.cluster.node | keys |
 ↪  length')
```

Handling missing/null values:

```
# Safe extraction - returns empty string if key missing or null
value=$(om daemon status -o json | jq -r '.path.to.key // empty')

# Check if value exists and is not null
if [ "$(om daemon status -o json | jq -r '.path.to.key // empty')"
 ↪  != "" ]; then
  echo "Value exists"
fi
```

9.30.3.2 Flat Format Parsing

Extract quoted string values:

```
# Pattern: key = "value"
value=$(om daemon status --output flat | grep
 ↪  "^cluster\.node\.ips\.status\.agent = " | cut -d'"' -f2)
```

Extract numeric values:

```
# Pattern: key = 123
value=$(om daemon status --output flat | grep
 ↪  "^cluster\.node\.ips\.status\.gen\.tch-001 = " | awk '{print
 ↪  $NF}')
```

Count occurrences:

```
# Count how many gen entries exist
count=$(om daemon status --output flat | grep -c
 ↪  "^cluster\.node\.tch-001\.status\.gen\.")
```

Check existence:

```
# Check if key exists
if om daemon status --output flat | grep -q
 ↪  "^cluster\.node\.tch-001\.status\.frozen_at = "; then
  echo "Node has frozen_at field"
fi
```

9.30.4 Performance Considerations

9.30.4.1 Single API Call Pattern

GOOD - Call once, extract multiple values:

```
# JSON approach
status=$(om daemon status -o json)
agent=$(echo "$status" | jq -r '.cluster.node.ips.status.agent')
gen=$(echo "$status" | jq -r '.cluster.node.ips.status.gen')
frozen=$(echo "$status" | jq -r
 ↪  '.cluster.node.ips.status.frozen_at')

# Flat approach
status=$(om daemon status --output flat)
agent=$(echo "$status" | grep "^cluster\.node\.ips\.status\.agent
 ↪  = " | cut -d'"' -f2)
gen_count=$(echo "$status" | grep -c
 ↪  "^cluster\.node\.ips\.status\.gen\.")
```

BAD - Multiple API calls for related data:

```
# Don't do this - 3 separate API calls
agent=$(om daemon status -o json | jq -r
 ↪  '.cluster.node.ips.status.agent')
gen=$(om daemon status -o json | jq -r
 ↪  '.cluster.node.ips.status.gen')
frozen=$(om daemon status -o json | jq -r
 ↪  '.cluster.node.ips.status.frozen_at')
```

9.30.4.2 Caching for Batch Operations

When validating multiple nodes, cache the cluster status:

```
# Get cluster status once
cluster_status=$(om daemon status -o json)

# Validate multiple nodes using cached status
for node in $nodes; do
  gen_data=$(echo "$cluster_status" | jq -r
   ↪  ".cluster.node.\"${node}\".status.gen // empty")
  if [ -n "$gen_data" ] && [ "$gen_data" != "null" ]; then
    echo "$node is reachable"
  fi
done
```

9.30.5 Format Comparison Example

Extracting the same information using both formats:

Task: Check if node tch-001 has heartbeat generation data

Using JSON:

```
gen_data=$(om daemon status -o json | jq -r
 ↪  '.cluster.node."tch-001".status.gen // empty')
if [ -n "$gen_data" ] && [ "$gen_data" != "null" ]; then
  gen_count=$(echo "$gen_data" | jq 'length')
  echo "Node has $gen_count heartbeat peers"
fi
```

Using Flat:

```
gen_count=$(om daemon status --output flat | grep -c
 ↪  "^cluster\.node\.tch-001\.status\.gen\.")
if [ $gen_count -gt 0 ]; then
  echo "Node has $gen_count heartbeat peers"
fi
```

Analysis: - JSON: More structured, easier to get complete gen object - Flat: Simpler for just counting, uses basic shell tools - Both are valid - choose based on what you need to extract

9.30.6 Error Handling

JSON with jq:

```
# Wrap in try-catch for jq errors
value=$(om daemon status -o json 2>/dev/null | jq -r '.path.to.key
 ↪  // empty' 2>/dev/null)
if [ $? -ne 0 ]; then
  echo "Failed to parse JSON"
```

```
  return 1
fi
```

Flat format:

```
# Check if om command succeeded
output=$(om daemon status --output flat 2>&1)
if [ $? -ne 0 ]; then
  echo "OpenSVC command failed: $output"
  return 1
fi
```

9.30.7 Prerequisites

Both JSON and flat parsing require basic tools:

Required for JSON: - `jq` - JSON processor (install via `apk add jq` on Alpine, `yum install jq` on Rocky)

Required for Flat: - `grep`, `cut`, awk - Standard POSIX tools (always available)

9.31 Node Health Validation

9.31.1 Node Reachability Detection (Official Method)

Implementation: Use `cluster.node.<n>.status.gen` object to determine node health.

Official Guidance from OpenSVC Team:
"cluster.node..status.gen is reliable"

What is status.gen? The `status.gen` object contains generation counters for heartbeat communication with peer nodes. Each entry represents an active heartbeat channel with another node in the cluster.

Example - Healthy Node:

```
{
  "cluster": {
    "node": {
      "tch-001": {
        "status": {
          "gen": {
            "ips": 7854,
            "tch-001": 9852,
            "tch-002": 9818
          }
        }
      }
```

```
    }
  }
}
```

Example - Unreachable Node:

```
{
  "cluster": {
    "node": {
      "tch-002": {
        "status": {
          "gen": null
        }
      }
    }
  }
}
```

Or the entire `status.gen` key may be missing.

Implementation Logic:

```
# Get gen data for node
gen_data=$(om daemon status -o json | jq -r
 ↪  ".cluster.node.\"${node_name}\".status.gen // empty")

# Check if exists and has entries
if [ -z "$gen_data" ] || [ "$gen_data" = "null" ]; then
  # Node not reachable - no heartbeat data
  return 3
fi

# Count peer entries (optional, for logging)
gen_count=$(echo "$gen_data" | jq 'length')

# If we get here, node is reachable
```

Why This Works: - Nodes participating in cluster heartbeat will have gen counters for each peer - Missing or null gen data means the node is not sending/receiving heartbeats - This is the same data OpenSVC uses internally to determine node health

9.31.2 Frozen State Detection

Implementation: Check `cluster.node.<n>.status.frozen_at` timestamp value.

OpenSVC Internal Behavior: OpenSVC uses Go's `time.Time.IsZero()` to test frozen state internally.

Frozen State Values: - **Frozen**: `"2025-10-29T08:31:59.358321416Z"` (real timestamp) - **Not Frozen**: `"0001-01-01T00:00:00Z"` (Go zero-value for time.Time) - **Not Frozen**: Field absent/null (node never frozen)

Why Zero Value? This is intentional Go idiom. Rather than using nullable types, Go uses zero values. The `0001-01-01T00:00:00Z` timestamp is Go's zero value for `time.Time` and indicates "no time set" = not frozen.

Implementation:

```
# Extract frozen_at timestamp
frozen_at=$(om daemon status -o json | jq -r
 ↪  ".cluster.node.\"${node_name}\".status.frozen_at // empty")

# Check if frozen (excluding zero value and null)
if [ -n "$frozen_at" ] && [ "$frozen_at" != "null" ] && [
 ↪  "$frozen_at" != "0001-01-01T00:00:00Z" ]; then
  # Node is frozen
  return 4
fi
```

9.32 IPS Command Interface

9.32.1 Required Implementation

The IPS must implement the following command interface for VM configuration retrieval:

```
n_ips_command vm get_config vm_id=<identifier>
```

9.32.2 Response Format

Text-based key=value pairs, one per line:

```
name=test-vm-01
cpu_count=4
ram_mb=8192
provision_method=virt-install
disk_1_a=/dev/disk/by-path/ip-10.31.0.100:3260-iscsi-iqn.2025-01.local.hps:storage
disk_1_b=/dev/disk/by-path/ip-10.32.0.100:3260-iscsi-iqn.2025-01.local.hps:storage
disk_2_a=/dev/disk/by-path/ip-10.31.0.100:3260-iscsi-iqn.2025-01.local.hps:storage
disk_2_b=/dev/disk/by-path/ip-10.32.0.100:3260-iscsi-iqn.2025-01.local.hps:storage
vxlan_1000=br-vxlan-1000
vxlan_1001=br-vxlan-1001
title=Development Web Server
description=Customer A web application server
```

9.32.3 Configuration Keys

Required Keys: - `name` - VM name (GUID issued by IPS) - `cpu_count` - Number of virtual CPUs - `ram_mb` - RAM allocation in megabytes - `provision_method` - Provisioning method (v1: only “virt-install” supported)

Optional Keys: - `disk_N_a` - Primary path for disk N (iSCSI device path) - `disk_N_b` - Secondary path for disk N (for multipath redundancy) - `vxlan_NNNN` - Bridge name for VXLAN VNI NNNN - `title` - Human-readable VM title - `description` - VM description/purpose

Storage Rules: - VMs may have zero disks (diskless thin servers) - Each disk consists of one or two paths - disk_N_a is always the primary path - disk_N_b is optional secondary path from different storage host - Disks numbered sequentially: disk_1_a, disk_2_a, disk_3_a, etc.

Network Rules: - VMs may have zero VXLAN networks - VXLAN keys formatted as: vxlan_ - Value is the bridge name to attach VM interface to - Multiple VXLANs supported per VM

9.33 Function Specifications

9.33.1 Node Validation Functions

9.33.1.1 1. o_vm_validate_node

Purpose: Validate that a node exists in the cluster and is healthy for VM operations.

File: `/srv/hps-system/lib/functions.d/o_vm-functions.sh`

Signature:

`o_vm_validate_node <node_name>`

Parameters: - `node_name` - Node name to validate (required)

Behavior:

Step 1: Parameter Validation

```
1.1. Check node_name is not empty
     → If empty: Log error, return 1
```

Step 2: Node Existence Check

```
2.1. Call: om node ls
2.2. Check if node_name in output
     → If not found: Log error, return 2
```

Step 3: Node Reachability Check (Official Method)

```
3.1. Call: om daemon status -o json
3.2. Extract: .cluster.node.<node_name>.status.gen
```

```
3.3. If gen_data is empty or null:
     → Log: Node not reachable (no heartbeat gen data)
     → Return 3
3.4. Count peer entries: jq 'length'
3.5. Log: Node is reachable (<count> heartbeat peers)
```

Step 4: Node Frozen State Check

```
4.1. Extract: .cluster.node.<node_name>.status.frozen_at
4.2. If frozen_at exists AND != "null" AND != "0001-01-01T00:00:00Z":
     → Log: Node is frozen (frozen_at: <timestamp>)
     → Return 4
4.3. Log: Node is not frozen
```

Step 5: Validation Success

```
5.1. Log: Node validated successfully
5.2. Return 0
```

Dependencies: - `om node ls` - List cluster nodes - `om daemon status -o json` - Get structured cluster status - `jq` - JSON processor for parsing - `o_log` - Logging function

Logging: - Parameter error: `o_log "o_vm_validate_node: node_name is required" "err"` - Not in cluster: `o_log "Node ${node_name} not found in cluster" "err"` - Not reachable: `o_log "Node ${node_name} not reachable (no heartbeat gen data)" "err"` - Is frozen: `o_log "Node ${node_name} is frozen (frozen_at: ${frozen_at})" "err"` - Success: `o_log "Node ${node_name} validated successfully" "info"`

Returns: - **Exit Code**: - 0: Node is valid and healthy - 1: Parameter validation failure - 2: Node does not exist in cluster - 3: Node not reachable (no heartbeat gen data) - 4: Node is frozen

Example Usage:

```
if o_vm_validate_node "tch-001"; then
  echo "Node is healthy"
  o_vm_create "abc-123" "tch-001"
else
  exit_code=$?
  case $exit_code in
    2) echo "Node not in cluster" ;;
    3) echo "Node not reachable" ;;
    4) echo "Node is frozen" ;;
  esac
fi
```

Implementation Note: Uses official `status.gen` API confirmed by OpenSVC team as reliable for determining node health.

9.33.1.2 2. o_vm_validate_node_quiet

Purpose: Validate node without logging (for use in selection logic).

File: `/srv/hps-system/lib/functions.d/o_vm-functions.sh`

Signature:

`o_vm_validate_node_quiet <node_name>`

Parameters: - `node_name` - Node name to validate (required)

Behavior: - Same validation as `o_vm_validate_node` - No logging output - Useful for filtering node lists

Returns: - Same exit codes as `o_vm_validate_node` (0-4)

Example Usage:

```
for node in $(o_vm_get_nodes_by_tag "tch"); do
  if o_vm_validate_node_quiet "$node"; then
    available_nodes="${available_nodes} ${node}"
  fi
done
```

9.33.1.3 3. o_vm_get_healthy_nodes

Purpose: Filter a node list to only healthy nodes.

File: `/srv/hps-system/lib/functions.d/o_vm-functions.sh`

Signature:

`o_vm_get_healthy_nodes <node_list>`

Parameters: - `node_list` - Space-separated list of node names (required)

Behavior: 1. Validate parameter not empty → return 1 if empty 2. For each node in list: - Call `o_vm_validate_node_quiet` - If returns 0, add to healthy_nodes list 3. Log count of healthy vs total nodes 4. Output healthy nodes to stdout 5. Return 0

Returns: - **Exit Code**: - 0: Success (outputs healthy nodes to stdout, even if empty) - 1: Parameter validation failure - **Stdout**: Space-separated list of healthy node names

Example Usage:

```
all_nodes=$(o_vm_get_nodes_by_tag "tch")
healthy_nodes=$(o_vm_get_healthy_nodes "$all_nodes")
if [ -n "$healthy_nodes" ]; then
  node=$(echo "$healthy_nodes" | awk '{print $1}')
  o_vm_create "abc-123" "$node"
fi
```

9.33.2 VM Orchestration Functions

9.33.2.1 4. o_vm_create

Purpose: Create and start a VM on specified TCH node using transient OpenSVC service.

File: `/srv/hps-system/lib/functions.d/o_vm-functions.sh`

Signature:

```
o_vm_create <vm_identifier> <target_node>
```

Parameters: - `vm_identifier` - Unique VM identifier (GUID) (required) - `target_node` - TCH node name (e.g., "tch-001") (required)

Behavior:

Step 1: Parameter Validation

```
1.1. Check parameter count == 2
     → If not: Log error, return 1
1.2. Check vm_identifier is not empty
     → If empty: Log error, return 1
1.3. Check target_node is not empty
     → If empty: Log error, return 1
```

Step 2: Validate Target Node

```
2.1. Log: Validating target node
2.2. Call: o_vm_validate_node "${target_node}"
2.3. Capture: validate_result=$?
2.4. If validate_result != 0:
     → Case validate_result:
       2: Log "Target node not found in cluster", return 2
     3: Log "OpenSVC daemon not running on target node", return 3
      4: Log "Target node is frozen or in error state", return 4
       *: Log "Node validation failed", return 2
```

Step 3: Log Operation Start

```
3.1. Log: "Creating VM ${vm_identifier} on node ${target_node}"
```

Step 4: Define Service Name

```
4.1. service_name="vm-ops-create-${vm_identifier}"
```

Step 5: Create OpenSVC Service with Task

```
5.1. Log: Creating task service (nodes: ips ${target_node})
5.2. Call: o_task_create "${service_name}" "create" "n_vm_create ${vm_identifier}" "i
5.3. Capture: create_result=$?
5.4. If create_result != 0:
     → Log: "Failed to create task service"
```

```
     → Return 5
```

Step 6: Wait for Instance Availability on Target Node

```
6.1. Log: Waiting for service instance (max 30s)
6.2. max_wait=30, waited=0, instance_ready=false
6.3. While waited < max_wait:
     6.3.1. Check: om instance ls | grep target_node
     6.3.2. If found:
            → instance_ready=true
            → Log: Instance available (${waited}s)
            → Break
     6.3.3. Sleep 1
     6.3.4. Increment waited
6.4. If instance_ready == false:
     → Log: Timeout waiting for instance
     → Call: o_task_delete "${service_name}"
     → Return 6
```

Step 7: Execute the Task

```
7.1. Log: Executing VM creation task
7.2. Call: o_task_run "${service_name}" "create" "${target_node}"
7.3. Capture: run_result=$?
7.4. If run_result != 0:
     → Log: Failed to execute VM creation
     → (Continue to cleanup)
```

Step 8: Delete the Service

```
8.1. Log: Cleaning up task service
8.2. Call: o_task_delete "${service_name}"
8.3. Capture: delete_result=$?
8.4. If delete_result != 0:
     → Log: Failed to cleanup service
```

Step 9: Determine Return Code

```
9.1. If run_result != 0 AND delete_result != 0:
    → Log: "VM creation failed AND service cleanup failed (exceptional state)"
     → Log: "Manual cleanup required: om ${service_name} purge"
      → Return 9
9.2. If run_result != 0 AND delete_result == 0:
     → Log: "VM creation failed (service cleaned up)"
     → Return 7
9.3. If run_result == 0 AND delete_result != 0:
    → Log: "VM created successfully but service cleanup failed (exceptional state)"
     → Log: "Manual cleanup required: om ${service_name} purge"
```

```
     → Return 8
9.4. If run_result == 0 AND delete_result == 0:
     → Log: "Successfully created VM"
     → Return 0
```

Dependencies: - `o_vm_validate_node` - Validate node health - `o_task_create` - Create OpenSVC service with task - `o_task_run` - Execute task on target node - `o_task_delete` - Delete OpenSVC service - `o_log` - System logging - `n_vm_create` - Node-side VM creation function (called by task)

Logging: - All operations logged with appropriate severity - Exceptional states clearly marked - Manual cleanup commands provided when needed

Returns: - **Exit Code**: - 0: Complete success (VM created, service cleaned) - 1: Parameter validation failure - 2: Target node not in cluster - 3: Target node daemon not running/reachable - 4: Target node frozen or in error state - 5: Task service creation failure - 6: Instance availability timeout - 7: Task execution failure (VM not created, service cleaned) - 8: Task succeeded but cleanup failed (EXCEPTIONAL - VM created, orphaned service) - 9: Task failed AND cleanup failed (EXCEPTIONAL - VM not created, orphaned service)

Exceptional States (codes 8, 9): - Indicate inconsistent system state - Require manual investigation - Service may need manual deletion: `om vm-ops-create-<vm_id> purge` - Check OpenSVC daemon logs for cleanup failure reason

Important Notes: - IPS is included in service nodes for management/cleanup only - Task always executes on target_node, never on IPS - Service nodes format: "ips ${target_node}" - 30-second timeout for instance propagation to target node

Example Usage:

```
# Basic usage with automatic validation
if o_vm_create "abc-123-def" "tch-001"; then
  echo "VM created successfully"
else
  exit_code=$?
  case $exit_code in
    2|3|4) echo "Target node not healthy" ;;
    5) echo "Failed to create service" ;;
    6) echo "Timeout waiting for service" ;;
    7) echo "VM creation failed" ;;
    8|9) echo "CRITICAL: Orphaned service requires manual cleanup"
     ↪  ;;
  esac
fi

# With node selection
vm_id="abc-123-def"
node=$(o_vm_select_node 4 8192)
if [ $? -eq 0 ]; then
```

```
  o_vm_create "$vm_id" "$node"
fi
```

9.34 Node Function Requirements

These functions are implemented in `/srv/hps-system/lib/node-functions.d/common.d/n_vm-functio`

9.34.1 n_vm_create

Status: ☒ Implemented and Tested

Signature:

`n_vm_create <vm_identifier> [title] [description]`

Purpose: Create VM on TCH node using virt-install

Behavior: 1. Call `n_ips_command vm get_config vm_id=${vm_identifier}` to fetch configuration 2. Parse key=value response into variables 3. Validate required fields: name, cpu_count, ram_mb, provision_method 4. Verify provision_method == “virt-install” (only supported method in v1) 5. Build virt-install command with: - All disk_N_a paths as –disk arguments - All disk_N_b paths as additional –disk arguments (for multipath) - All vxlan_NNNN as –network bridge= arguments - Optional –description with title/description fields 6. Execute virt-install command 7. Log results via `n_remote_log` 8. Return 0 on success, non-zero on failure

Required for: o_vm_create task execution

9.34.2 n_vm_start

Status: ☒ Implemented and Tested

Signature:

`n_vm_start <vm_name>`

Purpose: Start a stopped VM

Behavior: 1. Validate vm_name parameter 2. Execute: `virsh start "${vm_name}"` 3. Log result 4. Return 0 on success, 1 on failure

---

9.34.3 n_vm_stop

Status: ☒ Implemented and Tested

Signature:

`n_vm_stop <vm_name> [force]`

Purpose: Stop a running VM (graceful or forced)

Parameters: - `vm_name` - VM name (required) - `force` - If set to “force”, use destroy instead of shutdown (optional)

Behavior: 1. Validate vm_name parameter 2. If force == “force”: - Execute: `virsh destroy "${vm_name}"` - Log: Force stop 3. Else: - Execute: `virsh shutdown "${vm_name}"` - Log: Graceful shutdown 4. Return 0 on success, 1 on failure

---

9.34.4 n_vm_pause

Status: ☒ Implemented and Tested

Signature:

`n_vm_pause <vm_name>`

Purpose: Pause/suspend a running VM

Behavior: 1. Validate vm_name parameter 2. Execute: `virsh suspend "${vm_name}"` 3. Log result 4. Return 0 on success, 1 on failure

---

9.34.5 n_vm_unpause

Status: ☒ Implemented and Tested

Signature:

`n_vm_unpause <vm_name>`

Purpose: Resume a paused VM

Behavior: 1. Validate vm_name parameter 2. Execute: `virsh resume "${vm_name}"` 3. Log result 4. Return 0 on success, 1 on failure

---

9.34.6 n_vm_destroy

Status: ☒ Implemented and Tested

Signature:

`n_vm_destroy <vm_name>`

Purpose: Completely remove a VM (stop and undefine)

Behavior: 1. Validate vm_name parameter 2. Execute: `virsh destroy "${vm_name}"` (ignore errors if not running) 3. Execute: `virsh undefine`

`"${vm_name}" --remove-all-storage` 4. Log result 5. Return 0 on success, 1 on failure

9.35 Test Specifications

9.35.1 Test File

Location: `/srv/hps-system/scripts/tests/test_o_vm_create.sh`

Status: ☒ Implemented and All Tests Passing

Purpose: Validate all VM provisioning functions before integration

9.35.2 Test Categories

9.35.2.1 Unit Tests - Parameter Validation

```
test_o_vm_create_no_params              # No parameters
test_o_vm_create_one_param              # Missing second parameter
test_o_vm_create_empty_vm_id            # Empty vm_identifier
test_o_vm_create_empty_target_node      # Empty target_node
```

9.35.2.2 Node Validation Tests

```
test_o_vm_validate_node_basic           # Parameter validation
test_o_vm_validate_node_nonexistent     # Non-existent node
test_o_vm_validate_node_healthy         # Healthy node detection
test_o_vm_get_healthy_nodes             # Filter node lists
```

9.35.2.3 Integration Tests - Node Validation

```
test_o_vm_create_with_invalid_node      # Non-existent node
 ↪  rejection
test_o_vm_create_with_unhealthy_node    # Unhealthy node handling
```

9.35.2.4 Integration Tests - Full Lifecycle

```
test_o_vm_create_service_lifecycle      # Complete workflow test
```

9.35.3 Test Execution

Prerequisites: - Run on IPS - Source main function library: `source /srv/hps-system/lib/functions.sh` - OpenSVC cluster running with at least one healthy node - n_vm_create function deployed to TCH nodes - `jq` installed for JSON parsing

Execution:

```
cd /srv/hps-system/scripts/tests
./test_o_vm_create.sh
```

Expected Output:

```
========================================
o_vm_create Function Tests
========================================
Test Configuration:
  Target Node: tch-001
  Test VM ID: test-vm-1761734324
Prerequisites Check:
✓ o_vm_create function available
✓ o_vm_validate_node function available
✓ OpenSVC available
✓ Cluster has 3 node(s)
========================================
Unit Tests - Parameter Validation
========================================
✓ o_vm_create with no parameters should return 1
✓ o_vm_create with one parameter should return 1
✓ o_vm_create with empty vm_identifier should return 1
✓ o_vm_create with empty target_node should return 1
========================================
Node Validation Tests
========================================
✓ o_vm_validate_node with no parameters should return 1
✓ o_vm_validate_node with empty parameter should return 1
✓ o_vm_validate_node should return 2 for non-existent node
✓ o_vm_validate_node detects healthy nodes
✓ o_vm_get_healthy_nodes filters correctly
========================================
Integration Tests - Node Validation
========================================
✓ o_vm_create should return 2 for non-existent node
✓ No service should be created for invalid node
✓ o_vm_create should fail for unhealthy node
========================================
Integration Tests - Full Lifecycle
========================================
✓ o_vm_create validates node before operations
========================================
Test Summary
```

```
========================================
Tests run:    15
Tests passed: 15
Tests failed: 0

Result: PASS
```

9.36 Implementation Status

9.36.1 Phase 1: Node Validation Functions ☒ COMPLETE

- ☒ Implement `o_vm_validate_node` with JSON output parsing
- ☒ Use official `status.gen` API (confirmed by OpenSVC team)
- ☒ Implement `o_vm_validate_node_quiet`
- ☒ Implement `o_vm_get_healthy_nodes`
- ☒ Handle Go zero-value timestamps for frozen state
- ☒ Comprehensive logging and error handling

9.36.2 Phase 2: VM Orchestration Functions ☒ COMPLETE

- ☒ Implement `o_vm_create` with node validation
- ☒ Add instance availability wait logic (30s timeout)
- ☒ Service nodes include IPS for management
- ☒ Proper cleanup even on failure
- ☒ Detailed exit codes (0-9)

9.36.3 Phase 3: Node Lifecycle Functions ☒ COMPLETE

- ☒ Implement `n_vm_create` with virt-install
- ☒ Implement `n_vm_start`
- ☒ Implement `n_vm_stop` (graceful and force)
- ☒ Implement `n_vm_pause`
- ☒ Implement `n_vm_unpause`
- ☒ Implement `n_vm_destroy`

9.36.4 Phase 4: Test Implementation ☒ COMPLETE

- ☒ Create comprehensive test suite
- ☒ Parameter validation tests
- ☒ Node validation tests
- ☒ Integration tests
- ☒ All tests passing

9.36.5 Phase 5: Documentation ☒ COMPLETE

- ☒ Update specification with implementation details
- ☒ Document OpenSVC JSON and flat output usage
- ☒ Document official status.gen API
- ☒ Document Go zero-value timestamp handling
- ☒ Exit code reference table
- ☒ Comprehensive parsing best practices guide
- ☒ Troubleshooting guide

9.37 Next Steps (Future Enhancements)

9.37.1 Phase 6: IPS Command Interface (TODO)

- ☐ Implement `n_ips_command vm get_config` (replace mock)
- ☐ Design VM configuration storage on IPS
- ☐ Implement VM registry/database

9.37.2 Phase 7: Additional VM Operations (TODO)

- ☐ `o_vm_stop` - Orchestrated VM stop
- ☐ `o_vm_destroy` - Orchestrated VM removal
- ☐ `o_vm_list` - List all VMs across cluster
- ☐ `o_vm_status` - Get VM state from nodes
- ☐ `o_vm_migrate` - Move VM between nodes

9.37.3 Phase 8: Enhanced Node Selection (TODO)

- ☐ Capacity-aware node selection (CPU, RAM, VM count)
- ☐ Load balancing algorithm
- ☐ Resource reservation system
- ☐ Node affinity/anti-affinity rules

9.37.4 Phase 9: Advanced Features (TODO)

- ☐ VM templates and cloning
- ☐ Network boot (PXE) VMs
- ☐ VM snapshot management
- ☐ Live migration support
- ☐ HA and failover policies

9.38 Exit Code Reference

9.38.1 o_vm_validate_node Exit Codes

Code	Meaning	Action Required
0	Node healthy	None - proceed with operations
1	Invalid parameters	Fix function call
2	Node not in cluster	Verify node name, check cluster membership
3	Node not reachable	Check node daemon, network connectivity, heartbeat
4	Node frozen	Unfreeze node: `om node unfreeze --node <n>`

9.38.2 o_vm_create Exit Codes

Code	Meaning	VM State	Service State	Action Required
0	Complete success	Created	Cleaned	None
1	Invalid parameters	Not created	Not created	Fix parameters
2	Node not in cluster	Not created	Not created	Check node name
3	Node not reachable	Not created	Not created	Check node daemon
4	Node frozen	Not created	Not created	Unfreeze node

Code	Meaning	VM State	Service State	Action Required
5	Service creation failed	Not created	Not created	Check OpenSVC
6	Instance timeout	Not created	Cleaned	Check cluster health, may need longer timeout
7	Execution failed	Not created	Cleaned	Check n_vm_create logs on node
8	VM created, cleanup failed	**Created**	**Orphaned**	**Manual cleanup: `om <service> purge`**
9	Execution & cleanup failed	Not created	**Orphaned**	**Manual cleanup: `om <service> purge`**

Exceptional States (codes 8 and 9): - Indicate inconsistent system state - Require manual investigation - Check OpenSVC daemon logs: `om daemon logs` - Manual service cleanup: `om vm-ops-create-<vm_id> purge` - Verify VM state on target node: `virsh list --all`

9.39 Troubleshooting Guide

9.39.1 Node Validation Failures

Node not in cluster (exit code 2):

```
# List cluster nodes
om node ls

# Add node to cluster (if needed)
om node register <node_name>
```

Node not reachable (exit code 3):

```
# Check node status.gen from IPS
om daemon status -o json | jq
 ↪  ".cluster.node.\"${node_name}\".status.gen"
# Should return object with peer entries like: {"ips": 7854,
 ↪  "tch-001": 9852}
# If null or empty, node has no heartbeat

# On the node itself, check daemon
ps aux | grep opensvc
systemctl status opensvc  # or rc-service opensvc status

# Check network connectivity
ping <node_name>
ssh <node_name>

# Verify heartbeat configuration
om daemon status | grep -A5 "hb#"
```

Node frozen (exit code 4):

```
# Check frozen state
om daemon status -o json | jq
 ↪  ".cluster.node.\"${node_name}\".status.frozen_at"
# If timestamp is not "0001-01-01T00:00:00Z", node is frozen

# Unfreeze node
om node unfreeze --node <node_name>

# Verify unfrozen (should return zero value)
om daemon status -o json | jq
 ↪  ".cluster.node.\"${node_name}\".status.frozen_at"
# Should show: "0001-01-01T00:00:00Z"
```

9.39.2 Service Creation Issues

Service creation fails (exit code 5):

```
# Check OpenSVC daemon status
om daemon status

# Check IPS logs
tail -f /srv/hps-system/log/rsyslog/ips/$(date +%Y-%m-%d).log
```

```
# Try creating service manually
om vm-ops-test create --kw nodes="ips tch-001" --kw orchestrate=ha
```

Instance timeout (exit code 6):

```
# Check service status
om vm-ops-create-<vm_id> print status

# Check if service exists on target node
ssh <target_node> "om svc ls | grep vm-ops"

# Manually delete orphaned service
om vm-ops-create-<vm_id> purge
```

9.39.3 VM Creation Failures

Task execution fails (exit code 7):

```
# Check node logs
tail -f /srv/hps-system/log/rsyslog/<node_ip>/$(date
 ↪  +%Y-%m-%d).log

# Check if virt-install is installed on node
ssh <node> "which virt-install"

# Check if libvirt daemon is running
ssh <node> "rc-service libvirtd status"

# Test n_vm_create manually on node
ssh <node>
source /usr/local/lib/hps-bootstrap-lib.sh
hps_load_node_functions
n_vm_create "test-vm-id"
```

9.39.4 Orphaned Services (Exceptional States)

Exit codes 8 or 9 indicate orphaned services requiring manual cleanup:

```
# List all services
om svc ls

# Check service status
om vm-ops-create-<vm_id> print status

# Force delete service
om vm-ops-create-<vm_id> purge --force

# If purge fails, use daemon delete-config
om daemon delete-config --path /root/svc/vm-ops-create-<vm_id>
```

```
# Verify cleanup
om svc ls | grep vm-ops-create
```

9.40 File Structure

```
/srv/hps-system/
├── lib/
│   ├── functions.d/
│   │   ├── o_opensvc-task-functions.sh  (existing - task management)
│   │   └── o_vm-functions.sh            (NEW - VM orchestration + validation)
│   └── node-functions.d/
│       └── common.d/
│       └── n_vm-functions.sh        (NEW - node-side VM lifecycle)
└── scripts/
    └── tests/
     └── test_o_vm_create.sh         (NEW - comprehensive test suite)
```

9.41 Performance Considerations

9.41.1 OpenSVC API Call Optimization

Single Call Pattern (RECOMMENDED):

```
# Call once, extract multiple values
status=$(om daemon status -o json)
gen=$(echo "$status" | jq -r
 ↪  ".cluster.node.\"${node}\".status.gen")
frozen=$(echo "$status" | jq -r
 ↪  ".cluster.node.\"${node}\".status.frozen_at")
```

Multiple Call Anti-Pattern (AVOID):

```
# Don't do this - wastes API calls
gen=$(om daemon status -o json | jq -r
 ↪  ".cluster.node.\"${node}\".status.gen")
frozen=$(om daemon status -o json | jq -r
 ↪  ".cluster.node.\"${node}\".status.frozen_at")
```

9.41.2 Node Validation Caching

For operations validating multiple nodes, cache the cluster status:

```
# Get all node status once
cluster_status=$(om daemon status -o json)
```

```
# Validate multiple nodes using cached status
for node in $nodes; do
  gen_data=$(echo "$cluster_status" | jq -r
   ↪  ".cluster.node.\"${node}\".status.gen // empty")
  if [ -n "$gen_data" ] && [ "$gen_data" != "null" ]; then
    echo "$node is reachable"
  fi
done
```

9.42 Security Considerations

9.42.1 VM Identifier Validation

VM identifiers are used in: - Service names: `vm-ops-create-${vm_identifier}` - OpenSVC commands: `om vm-ops-create-${vm_identifier} ...`

Recommendations: - Use GUIDs/UUIDs for VM identifiers - Validate format before use (alphanumeric + hyphens only) - Avoid user-supplied strings without validation

9.42.2 Node Name Validation

Node names from user input should be validated:

```
# Ensure node name matches cluster nodes
if ! om node ls | grep -qx "${node_name}"; then
  # Reject invalid node name
fi
```

9.42.3 Command Injection Prevention

All variables used in shell commands are properly quoted:

```
# GOOD - prevents injection
o_task_create "${service_name}" "create" "n_vm_create
 ↪  ${vm_identifier}" "ips ${target_node}"

# BAD - vulnerable to injection
o_task_create $service_name create "n_vm_create $vm_identifier"
 ↪  "ips $target_node"
```

9.43 Specification Sign-off

Specification Version: 2.1
Implementation Status: COMPLETE

Test Status: ALL TESTS PASSING

Date: 2025-10-29

Key Achievements: - ✅ Full node validation using official OpenSVC `status.gen` API - ✅ Comprehensive parsing guide for JSON and flat output formats - ✅ Robust VM orchestration with health checks and timeout handling - ✅ Complete node-side VM lifecycle functions - ✅ Comprehensive test suite with 15/15 tests passing - ✅ Production-ready error handling and logging - ✅ Proper handling of OpenSVC internals (Go zero-value timestamps) - ✅ Performance optimization guidance for API calls

Ready for: Production deployment with mock VM configuration data. Real VM provisioning pending IPS `vm get_config` command implementation.

9.44 Notes

- All functions follow HPS naming conventions (o_ prefix for IPS, n_ prefix for nodes)
- All functions include comprehensive documentation headers
- Error handling follows fail-fast principle with detailed exit codes
- Logging uses appropriate severity levels
- Functions are modular and independently testable
- Implementation uses existing OpenSVC task infrastructure
- Design supports future enhancements without breaking changes
- OpenSVC JSON output format used for complex data extraction (status.gen)
- OpenSVC flat output format remains viable for simple single-value extraction
- Node validation uses official API confirmed by OpenSVC development team
- Service cleanup handles all failure scenarios including exceptional states
- Requires `jq` for JSON parsing (standard tool, widely available)