OPEN DIGITAL
CONSULTING CO

DRAFT Platform documentation

HOX project

Stuart J Mackintosh

Tuesday 18 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.

1.1 Introduction

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

2 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.

2.1 Booting a host

Stub: PXE boot process overview and selecting a host profile.

2.2 First cluster setup

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

2.3 Installation

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

2.4 Prerequisites

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

2.5 Verification

Stub: Confirming services are active and hosts are provisioned correctly.

3 Installing HPS

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

3.1 Hardware

Depends on the scenario, for example:

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

3.2 Obtaining HPS

How to acquire HPS source

- HPS Repo link
- Link to ISO downloads

3.3 Dependencies and prerequisites

- Operating system ISO's

3.4 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.

3.5 Service verification

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

3.6 Upgrades and maintenance

Stub: Keeping `hps-system` updated without overwriting configuration files.

4 Deploying and configuring a cluster with nodes

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

4.1 Cluster configuration

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

4.2 Cluster creation

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

4.3 Distribution management

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

4.4 Host profiles

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

4.5 Node provisioning

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

4.6 Service management

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

4.7 Verification

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

5 System reference

Static technical reference information for HPS, including function documentation, troubleshooting guides, and configuration details.

5.1 Keysafe Token Authentication Flow

5.1.1 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.

5.1.2 Flow Diagram

5.1.3 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

5.1.4 Implementation Notes

- The Client Wrapper and DRH Wrapper handle the secure token exchange

Authorisation sequence

Figure 5.1: Authorisation sequence

- The actual data transfer happens via rsync after authentication
- Token consumption prevents replay attacks
- Short TTL (60 seconds) limits exposure window

5.2 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.

5.2.1 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‘

Thing you want to set	Where / How
Distribute service configs to other nodes	`om mysvc push` / `om mysvc sync`

5.2.1.1 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

5.2.1.2 References

5.3 Environment variables

Stub: Explanation of exported variables and their purpose.

5.4 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.

5.5 OpenSVC V3 Alpha Cheatsheet

Version tested: v3.0.0-alpha87

5.5.1 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
```

5.5.2 Service Management

5.5.2.1 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"
```

5.5.2.2 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
```

5.5.3 Task Resources

5.5.3.1 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"
```

5.5.3.2 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
```

5.5.3.3 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>
```

5.5.4 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
```

5.5.5 Sync Resources

5.5.5.1 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”.

5.5.6 Configmaps

5.5.6.1 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
```

5.5.6.2 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

5.5.7 Naming Conventions

5.5.7.1 Valid Names

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

5.5.7.2 Object Path Formats

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

- Volumes: `vol/<name>` (assumed, not tested)

5.5.8 Resource Section Naming

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

5.6 Working Patterns

5.6.1 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
```

5.6.2 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
```

5.6.3 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>
```

5.6.4 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"
```

5.7 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

5.8 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
```

5.9 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.

5.10 Paths

5.10.1 hps provisioning node

5.10.2 hps operating node (TCN, CCN etc)

Storage under /srv/storage

scripts under /srv/scripts

/srv is iscsi mounted

everything else is transient.

5.11 Reference configurations

Stub: Example `cluster.conf`, `host.conf`, and service configuration files.

6 Storage Provisioning

6.1 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.

6.2 Prerequisites

All operations require the node functions to be sourced first:

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

6.3 Architecture

6.3.1 Node Types

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

6.3.2 Components

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

6.4 Manual Operations

6.4.1 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
```

6.4.2 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

6.4.3 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

6.5 Low-Level Operations

6.5.1 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
```

6.5.2 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
```

6.6 OpenSVC Integration

6.6.1 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
```

6.6.2 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>
```

6.7 Troubleshooting

6.7.1 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

6.7.2 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
   ```

6.8 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
```

6.9 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

6.10 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`

6.11 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).

6.11.1 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.

6.11.2 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.

6.11.3 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.

6.11.4 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.

6.11.5 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.

6.11.6 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.

6.12 Troubleshooting

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

6.12.1 logging

See the log files / syslog

6.12.2 Function test

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

7 Functional Reference

7.0.1 `_apkovl_create_modloop_setup`

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

Function signature: 50db504474837c5cc2cca18c3ded5c2d55d20f9fcbf7de3fb45613840ce58d3d

7.0.2 Function Overview

The function `_apkovl_create_modloop_setup()` sets up a modloop for Alpine Linux. Its main tasks are constructing a local directory, defining the modules to load, creating a modloop setup script that downloads, mounts, and loads kernel modules, and finally confirming that the setup script is executable. It accepts three arguments representing the base directory, the gateway IP, and the Alpine version.

7.0.3 Technical Description

- **Name**: `_apkovl_create_modloop_setup`
- **Description**: This function sets up a modloop by creating a local directory, defining the modules to load, creating a setup script that downloads the kernel modules, mounts them, and loads them.
- **Globals**: No global variables are used directly, but function calls to `hps_log` may use predefined globals within that function.
- **Arguments**:
  - `$1` (base_dir): Base directory where the local.d directory will be created
  - `$2` (gateway_ip): Gateway IP address used in MODLOOP_URL
  - `$3` (alpine_version): Version of Alpine Linux used in MODLOOP_URL
- **Outputs**: Logs (debug and error level) generated by `hps_log`, and a modloop setup script placed in the local.d directory.
- **Returns**: Returns 1 if the local.d directory or the setup script couldn't be created, otherwise returns 0.
- **Example usage**:

```
_apkovl_create_modloop_setup "/home/user/directory"
 ↪  "192.168.0.1" "3.11"
```

7.0.4 Quality and Security Recommendations

1. Make sure argument validation is in place. This function already uses bash's null variable `-?` option to ensure the required arguments are provided.
2. Add more descriptive error messages in `hps_log` calls for better troubleshooting.
3. Make sure permissions for newly created directory and setup script are properly set to ensure security.
4. Use HTTPS instead of HTTP in URL for modloop download to add an extra layer of security.
5. Integrate logging into a central solution to identify potential security threats or malpractices.
6. Check the existence of directories and files before trying to create or modify them.
7. Before downloading the modloop, consider adding a checksum validation to confirm file integrity.

7.0.5 `_apkovl_create_resolv_conf`

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

Function signature: 7f68c373f65cdb9fd367f8e973f0778486a609eeeb348c3d3779336610f77261

7.0.6 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.

7.0.7 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"
```

7.0.8 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.

7.0.9 `_apkovl_create_runlevel_config`

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

Function signature: b3eb801c841e82a67fe14755dc7ba5f92e58b30d32010b4aa81733983d6b0626

7.0.10 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.

7.0.11 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"
```

7.0.12 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).

7.0.13 `_apkovl_create_structure`

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

Function signature: de21a4a364feb0c001016f8d48f06a318c8c14aa1115120e71e72b590755ee99

7.0.14 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.

7.0.15 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
```

7.0.16 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.

7.0.17 `build_dhcp_addresses_file`

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

Function signature: e4e89283fe1d64f56e0de04c05dbae29598781f49b67ce46af38b86c9af58337

7.0.18 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.

7.0.19 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
```

7.0.20 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.

7.0.21 `build`

Contained in `lib/host-scripts.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: 56e136787268e5be7a6960ee47b56f75a2c85fab6db9a387f30a81584efcfb76

7.0.22 1. Function Overview

The `build()` function in Bash is a very basic one. Its primary role is to return a 0 status code. In Bash, a return code of 0 indicates a successful operation or execution. This means when the `build()` function is called, it simply returns without performing any operation, signaling that the command completed successfully.

7.0.23 2. Technical Description

Name: `build()`

Description: This is a simple Bash function that returns a status code of 0, indicating that the operation was successful.

Globals: None

Arguments: None

Outputs: No explicit output. Implicitly, returns status code 0.

Returns: Status code 0, indicating a successful operation.

Example usage:

```
source script.sh  # where build function is defined
build
echo $?  # This will output 0, showing a successful operation
```

7.0.24 3. Quality and Security Recommendations

1. Even though `build()` function does not contain any harmful operations, it's good to use code scanning tools for checking potential security lapses in complex scripts.
2. The function name `build` seems a bit misleading as it doesn't actually build anything.
3. Always ensure that Bash functions are well-documented and reviewed by peers for improving the code quality.
4. Adopt proper error handling mechanism rather than just a binary 0 or 1 output. A more verbose error message may prove helpful in more complex scripts.
5. Expand the functionality of this function or reconsider its necessity, as its current form add no tangible output or process.

7.0.25 `build_yum_repo`

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

Function signature: 55fba2752d840b45670dcc10f8be084d3234f9c87a37a1959fcde6a9247be828

7.0.26 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.

7.0.27 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"`

7.0.28 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.

7.0.29 `build_zfs_source`

Contained in `lib/node-functions.d/common.d/n_storage-functions.sh`

Function signature: 6c2e11707d627f033ec6c0466cd422c8739078f93ea989430f8a0a9c0187dac0

7.0.30 Function overview

The function `build_zfs_source` is particularly designed to operate as a default build in non-distro-specific environments. At runtime, it prints a log message stating its operation, then proceeds to announce the requirement for the local system config file to have an implemented ZFS build process. The function finally returns an exit status of 1, symbolizing an error or false condition.

7.0.31 Technical description

Name: `build_zfs_source`

Description: This function is the default builder for ZFS source code. It prioritizes the implementation of a local systems `config file to manage the building of ZFS. It prints a log message and returns1`' if the local system lacks a configured ZFS build process.

Globals: No globals are used directly within this function.

Arguments: This function does not take formal arguments.

Outputs: The string “Running default build_zfs_source (not distro-specific)” is logged. Furthermore, if the local system fails to have an implemented ZFS build process, the string “This system must implement its own ZFS build process through the local system config file.” is written to standard output.

Returns: The function will always return 1 (false).

Example usage:

```
build_zfs_source
```

7.0.32 Quality and security recommendations

1. Always ensure proper implementation of a ZFS build process in the local system config file. This function expects it and will run successfully only if the local system is equipped with it.
2. Since the function returns a consistent 1 exit status, it might not be entirely useful within scripts that rely heavily on successful exit statuses (0). Adjustments may need to be made to either have a successful exit status on successful execution or handle the 1 exit status accordingly within the scripts using this function.
3. Given the function logs a message each time it runs, make sure the logging system is appropriately configured to handle and store those logs, particularly for troubleshooting purposes.
4. The logging message could be made more specific as to what the function is actually doing – an unclear message might make debugging more difficult.
5. Be careful with echo statements, as they can occasionally present security vulnerabilities, such as Injection. Use printf rather than echo for more reliable and secure output.

7.0.33 `cgi_auto_fail`

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

Function signature: 6b67dd57145386562143b5a27ad4c4f0a1e5170fc073f8d4e91738ade5779a4d

7.0.34 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.

7.0.35 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"`

7.0.36 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.

7.0.37 `cgi_fail`

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

Function signature: 5bc8c57b97b640ef4a194c0065da11fec44b1f1bb0525bf46e8163d2a50cabd5

7.0.38 Function overview

This function `cgi_fail()` is used within a CGI script to handle errors. It takes an error message as an argument, `cfmsg`, logs this message to some error tracking system via the `hps_log()` function, and then responds to the HTTP request with a plain header and the same error message echoed back. It's an in-built function for managing errors in CGI scripts.

7.0.39 Technical description

- **Name:** cgi_fail

- **Description:** This function handles errors in CGI scripts by logging errors and sending responses with plain headers and echoed error messages.
- **Globals:** None.
- **Arguments:**
  - `$1: cfmsg` – Error message that will be logged and sent back as response.
- **Outputs:** The function outputs an error message.
- **Returns:** This function does not return anything.
- **Example Usage:** New error message can be passed directly to function as follows `bash cgi_fail "An error has occurred."`

7.0.40 Quality and security recommendations

1. Validate the input: You should consider validating your function's input. This way it ensures that the error message passed is a string so that it can be correctly processed.
2. Error handling and logging: You should ensure proper error handling and logging is in place. This is vital for auditing and rectifying the faults in the system. Implement a standardized error logging method.
3. Escape special characters: In the echoed error message, make sure to escape any HTML special characters for security purpose.
4. Usage of local variables: Use local variables where possible. Usage of global variables may lead to the risk of variable pollution.
5. Use appropriate HTTP status codes: Always use appropriate HTTP status codes in conjunction with error messages to help client understand the problem better.

7.0.41 `cgi_header_plain`

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

Function signature: cce7eb1544681966ec11d5f298135158497fc2ac56c89b00c63c84b8e1bc733a

7.0.42 Function Overview

The `cgi_header_plain` function is a Bash shell function that is designed to produce the HTTP header for a plain text response. In a CGI (Common Gateway Interface) context, this function can be used to clearly define the content type of the output as plain text.

7.0.43 Technical Description

```
def cgi_header_plain:
    - name : cgi_header_plain
    - description : A Bash function that prints the HTTP header for
 ↪  a plain text response, typically used in a CGI context.
    - globals : None
    - arguments : None
```

```
- outputs :
    - "Content-Type: text/plain"
    - A line break
- returns : None
- example usage :
    ```
    #!/bin/bash
    cgi_header_plain
    ```
```

7.0.44 Quality and Security Recommendations

1. Consider checking the status of the `echo` commands to ensure that they successfully sent the outputs.
2. If this function is used in a larger script, make sure that it's being called appropriately and that the output is being properly used.
3. Ensure secure execution with proper user privileges. Running the script with unnecessary admin rights can be a security risk.
4. Audit and keep a record of function usage in order to trace and troubleshoot any issues when they occur.
5. Include error handling to respond to failed operations and cleanly exit the script when necessary. This will prevent any misuse or security vulnerabilities.

7.0.45 `cgi_log`

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

Function signature: 9f2c2cf7c0d57e85a08611717b5d691eddf235f096bbc311bf9d58541f0c77b3

7.0.46 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.

7.0.47 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"`

7.0.48 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.

7.0.49 `cgi_param`

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

Function signature: 5007f95c313c04a01df7ba39bff0241f44511cd570d795bc11a890edf032f323

7.0.50 Function overview

The bash function `cgi_param` is designed to decode and retrieve parameters from the `QUERY_STRING` in a CGI context. The function uses a query-string from the CGI context and processes it to detect commands and key-value pairs. The function is triggered to process the query string only once, with subsequent calls to commands (get, exist, equals) retrieving or comparing the saved values. If an unknown command is passed, an error message is returned.

7.0.51 Technical description

- **Name**: `cgi_param`
- **Description**: This function parses a query string into parameters, then provides a way to retrieve a parameter value, check if a parameter exists, or see if a parameter's value matches a provided value by using the 'get', 'exists' or 'equals' commands respectively.
- **Globals**:
  - `QUERY_STRING`: The string to extract parameters and their values from.
  - `__CGI_PARAMS_PARSED`: Global flag to detect if the query string has been parsed.
  - `CGI_PARAMS`: An array to save decoded keys and their values.
- **Arguments**:

  - $1: The command to run: ‘get’, ‘exists’, ‘equals’.
  - $2: The name of the parameter.
  - $3: Optional. The value to compare against when the command is ‘equals’.
- **Outputs**: If the ‘get’ command is called, the value of the named parameter. If not, a success status or an error message in case of an invalid command.
- **Returns**: The function can return different values depending on the command given. If ‘get’ command, will print value to stdout. If ‘exists’, 0 is returned if parameter exists, 1 if not. If ‘equals’, returns 0 if parameter equals given value, 1 if not. Returns 2 if command is invalid.
- **Example usage**:

```
cgi_param get username
cgi_param exists page_count
cgi_param equals user_role admin
```

7.0.52 Quality and security recommendations

1. Use stricter validation on parameter keys while parsing. Right now, only alphanumeric characters plus underscores are allowed. However, consider more restrictive set.
2. Be sure to escape all variable expansions to prevent code injection.
3. Consider ways of handling or communicating parse failures more explicitly - it may prove difficult to debug if the `QUERY_STRING` is not formatted as expected.
4. Implement a more robust error handling mechanism. For example, when the user provides an invalid command, an exception should be handled that doesn’t allow further execution of the script.
5. Add more guard clauses for blank, null, or unexpected inputs to avoid unexpected behaviour.
6. Always try to keep your function’s behavior predictable and documented.

7.0.53 `cgi_success`

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

Function signature: 3c468a0d5bf1a1d432f4efcb2a108889f0d0e762f542f50c29b92350f02de3bc

7.0.54 Function overview

The `cgi_success` function is a part of CGI (Common Gateway Interface) scripting in bash. This function is designed to return a success message after successfully performing a CGI operation. It first calls another function, `cgi_header_plain`, likely designed to output some sort of standardized header, and then prints a message, which is provided as an argument.

7.0.55 Technical description

Name: `cgi_success`

Description: This function is part of CGI scripting in bash. It prints a plain header (via the `cgi_header_plain` function) and then a success message, which is provided as an argument when calling the function.

Globals: None.

Arguments: - `$1`: A string. The output message to be printed as a result of successful operation.

Outputs: The function outputs a plain header followed by the given success message.

Returns: Output is directed to STDOUT, there are no return values in the function.

Example usage:

```
cgi_success "The operation was completed successfully."
```

7.0.56 Quality and security recommendations

1. Input validation: Ensure that the input argument (`$1`) is being properly sanitized and validated. For example, it is checked for null values and potentially harmful characters that may cause unwanted actions when output.
2. Documentation: Each function should have a short comment explaining its purpose and any notable behaviors to aid future development and maintenance.
3. Error handling: Appropriate error handling procedures should be in place for when `cgi_header_plain` fails or generates an exception.
4. Provide a means for logging operations in order to aid debugging and provide transparency over the system's operations.
5. Coding standards: Follow good bash scripting practices, including usage of quotation marks around variable references and consistent indentation and naming conventions.
6. Secure practices: It's best not to display too much information in error messages, as this could potentially expose sensitive details which might be exploited by an attacker. The messages returned by this function should be reviewed for such concerns.

7.0.57 `check_and_download_latest_rocky`

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

Function signature: ac86f8c78c7149b4fbc3125a2f52dd65b160ad01f437b5eeb5b504bd1a90c6b1

7.0.58 Function overview

This shell function `check_and_download_latest_rocky`, written in bash script, is designed to check for the latest version of Rocky Linux for a specific architecture and

download the ISO file if it is missing. It sets up the target base directory and checks whether the latest version ISO file is present. If it is not present, it downloads the ISO from the base url which is initially defined.

7.0.59 Technical description

- **Name**: `check_and_download_latest_rocky`
- **Description**: This bash function checks and downloads the latest version of Rocky Linux for a specific architecture if it is missing.
- **Globals**: `HPS_DISTROS_DIR`: Directory for storing ISOs.
- **Arguments**: None.
- **Outputs**: Logs and messages about the process and final results.
- **Returns**: None in this function but could return 1 if erred in practical usage.
- **Example Usage**: `check_and_download_latest_rocky`. As it doesn't require any arguments, the function can be called without providing any.

7.0.60 Quality and security recommendations

1. **Validating inputs**: Input variables, especially those sourced from outside the function such as `HPS_DISTROS_DIR`, should be checked and validated to ensure they are correct and safe.
2. **Error trapping**: Consider adding more error trapping to handle situations where commands within the function fail.
3. **Use More Robust Code**: String operations on paths can be replaced with more robust code using `dirname` and `basename`.
4. **Use `https://`:** Always use secure protocol (https) while downloading files for ensured security.
5. **Checksum Verification:** For further security, the downloaded ISO could be verified against a known checksum to ensure its integrity. Any mismatches should trigger a warning or error.
6. **Permission and Access:** Ensure proper permission schemes for the directories and files mentioned in the function.

7.0.61 `check_available_space`

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

Function signature: 106f18dba6188914b70a8af6d1ddf900c6e5702a9f782fb29e86c3906bc878f6

7.0.62 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.

7.0.63 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.

7.0.64 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.

7.0.65 check_latest_version

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

Function signature: 3b874dd0168548a5363b9c63f357dab1016d452e1155213b1190930b50bb44c5

7.0.66 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.

7.0.67 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`

7.0.68 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.

7.0.69 `check_zfs_loaded`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: e83726e842477cf79c67cdcf1de046b56eacfafd7f97ba195d70b1f2bccfabc2

7.0.70 Function Overview

The `check_zfs_loaded` function checks whether the ZFS (Zettabyte File System) is installed and operational on the machine where the script is running. If the ZFS command isn't found or the ZFS kernel module isn't loaded, the function attempts to perform the necessary actions and gives feedback on the actions performed. The function ultimately returns 0 if the system passes all checks; otherwise it returns 1.

7.0.71 Technical Description

7.0.71.1 Name

`check_zfs_loaded`

7.0.71.2 Description

This bash function checks two conditions: if the ZFS command is found on the system, and if the ZFS kernel module is loaded. If any of these conditions is not met, it prompts the user with an error message. Moreover, if the ZFS module is not loaded, it attempts to load it using the `modprobe` function. This operation might require superuser privileges.

7.0.71.3 Globals

None.

7.0.71.4 Arguments

No arguments taken.

7.0.71.5 Output

The function outputs to stdout:

- A successful or unsuccessful message indicating ZFS command installation status.

- A status update on whether the ZFS module is loaded, whether a load attempt was made, and whether that was successful.

7.0.71.6 Returns

- `1`: if either the ZFS command is not found OR the ZFS module failed to load.
- `0`: if the ZFS command is found AND the ZFS module is successfully loaded.

7.0.71.7 Example Usage

```
if check_zfs_loaded; then
  echo "ZFS is ready to use."
else
  echo "ZFS is not properly set up."
fi
```

7.0.72 Quality and Security Recommendations

1. Authentication Check: The function should check if the user has required permissions to perform actions such as loading a kernel module using `modprobe`. Otherwise, it could misleadingly signal a failure when the underlying issue is a lack of permission.
2. Error Catching: It might be beneficial to add additional error handling or checking. For example, catching and further diagnosing errors that result from the `command` or `modprobe` functions could provide more clarity as to what caused a failure.
3. Reliable Checking: Checking presence of ZFS only through presence of `zfs` command and loading status of module might not fully confirm its operational status. Integrate a more reliable way of verifying whether ZFS is properly working, such as creating a test file on a ZFS filesystem.
4. Logging: Consider adding logging for better troubleshooting capabilities. Rather than just echoing the status, also write it into a log file.
5. Use Safe Bash Flags: Consider setting safe bash flags (`set -euo pipefail`) at the beginning of your scripts to avoid certain common bash pitfalls.

7.0.73 `cidr_to_netmask`

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

Function signature: f7b2c0eaf8bcc538e387c868ecc0741fec832f46b23a43a4bead09f5431bffb5

7.0.74 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.

7.0.75 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
```

7.0.76 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.

7.0.77 `cli_color`

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

Function signature: 5b1da9ceb66e5db9828e84c114d322c5fd60f95ee979773f19784a6dbb747c76

7.0.78 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.

7.0.79 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}"
```

7.0.80 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.

7.0.81 `cli_info`

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

Function signature: 28a3fbfdceeaf8bd49d31ad9ce80e2034af53e3e598755aaaa9e1bba6bb9cd30

7.0.82 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.

7.0.83 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"`

7.0.84 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.

7.0.85 cli_init_colors

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

Function signature: e0b1c767cca4df9b582c6ff8ad624787bdca5f882be1cc1fd7aa5eb774969d6b

7.0.86 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.

7.0.87 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 “$COLOR_REDThistextisred${COLOR_RESET}” “‘

7.0.88 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.

7.0.89 `cli_note`

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

Function signature: 1b1a001917f29b7c34dd816c043b9b720e5299789d848d1a2d0f19f67d867d1f

7.0.90 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.

7.0.91 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`

7.0.92 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.

7.0.93 `cli_prompt`

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

Function signature: f49e68ce40ea9bb18b1ae2b2549fbaf0b8bc3bef5f929a8a9764769dc4ce04d5

7.0.94 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.

7.0.95 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"
```

7.0.96 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.

7.0.97 `cli_prompt_yesno`

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

Function signature: f47ec2eb44075d167f6f356ea8bd4fc03757eb9a99d783f80e432210369044ff

7.0.98 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.

7.0.99 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.

7.0.100 `cli_select`

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

Function signature: c980cde4828d1b931c6014c0b115b19f394558bf05df9574129aae3aff155311

7.0.101 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.

7.0.102 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"
```

7.0.103 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.

7.0.104 `cli_set_active_cluster`

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

Function signature: 57af9df89602d50cd32305c1772eda685805080ffdb84fb6a47bc26ab7b1630d

7.0.105 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.

7.0.106 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"
```

7.0.107 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.

7.0.108 `cluster_config`

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

Function signature: 7945db2fcd65e8be1bffb4a38e55baa1da3c40817d5dbfa6899ec1986871997e

7.0.109 Function overview

The `cluster_config()` function is a bash function designed to retrieve, set, or confirm the existence of key-value pairs within a cluster configuration file. It takes four

arguments, three of which are optional, namely an operation to be executed (op - either get, set, or exists), a key (key), a value (value), and a cluster name (cluster). It utilizes local variables and command-line utilities such as grep, sed, cut to perform operations on the configuration file.

7.0.110 Technical description

- name: cluster_config()
- description: Function to get, set, or confirm the existence of keys and values in a specified or active cluster configuration file.
- globals: [HPS_CONFIG_DIR: Directory containing the cluster configuration files]
- arguments: [$1: The operation to be performed (get, set, or exists), $2: The key in the config file to be processed, $3: The value to be assigned to the key when performing set operation (optional), $4: The name of the cluster (optional)]
- outputs: Depending on the operation, it may output the value associated with a key (get operation), a confirmation message (set operation), or indication of the existence of a specific key (exists operation). It may also output error messages in case of failure.
- returns: The function can return an exit status of 1 or 2 indicating failure due to certain conditions.
- example usage:

  To get the value of a key in the active cluster:

  ```
  cluster_config get my_key
  ```

  To set the value of a key in a specific cluster:

  ```
  cluster_config set my_key my_value my_cluster
  ```

  To check if a key exists in the active cluster:

  ```
  cluster_config exists my_key
  ```

7.0.111 Quality and security recommendations

1. Consider documenting the function more systematically: Describing accepted arguments, operation types and errors in return values would improve maintenance and usability of the function.
2. The function should validate the input more carefully in order to prevent injection attacks. For example, it might process special characters in the key and value variables which could lead to unexpected behavior.

3. The number of global variables should be minimized. For instance, instead of relying on HPS_CONFIG_DIR, the function could be refined to allow passing of the directory path as an argument.
4. This function performs several commands without verifying their successful execution. These commands should be checked for success (and potential error messages handled) to prevent inconsistent or faulty output.
5. Hardcoded strings such as error messages could be replaced with constants or configuration options to improve consistency and maintainability.
6. Setting default values for optional parameters (namely, `value` and `cluster`) can help streamline the function's usage and avoid confusion.

7.0.112 `cluster_has_installed_sch`

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

Function signature: befb7e577a31bf8e64c1179ffa1c6cd4d2ec9a30913e1c6b26986c37fd0762cc

7.0.113 Function overview

This function, `cluster_has_installed_sch()`, checks a directory containing configuration files by reading and processing each file one at a time. Specifically, it checks for the presence of files ending with `.conf` extension. Within each file, it reads key-value lines and processes the lines containing "TYPE" and "STATE". It then checks if `type` is "SCH" and `state` is "INSTALLED". If it finds a match, it returns 0, indicating success. If no match is found after reading all the files, it returns 1, indicating failure.

7.0.114 Technical description

- **name**: cluster_has_installed_sch
- **description**: The function checks a directory of configuration files to find if a file contains TYPE=SCH and STATE=INSTALLED.
- **globals**: [HPS_HOST_CONFIG_DIR: The directory containing the configuration files (.conf) to be checked.]
- **arguments**: None.
- **outputs**: Outputs nothing.
- **returns**: Returns `0` if a match is found; otherwise it returns `1`.
- **example usage**:

```
$ cluster_has_installed_sch
$ echo $?
0
```

7.0.115 Quality and security recommendations

1. Add a check at the beginning of the function to ensure HPS_HOST_CONFIG_DIR is set and is a valid directory.
2. Provide a graceful exit or report an error message if the configuration directory does not exist.
3. Consider using stricter checks when processing each file's content. This can help prevent potential issues related to unexpected data.
4. Add more comments throughout the script to explain the function of each part.
5. In terms of security, be aware of potential "Path Traversal" vulnerabilities if the directory contains symbolic links pointing outside of the intended directory tree. You might want to add a check to ensure symbolic links are not processed.

7.0.116 `cluster_storage_init_network`

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

Function signature: a2658772004788852879 3a67b36d48651fe7668e31b2136db1f521752f8ebd7d

7.0.117 Function overview

The `cluster_storage_init_network` function sets up the initial configuration of a network for a cluster storage. It retrieves the number of storage networks to be configured, validates inputs, and then sets the configurations accordingly. It repeats the process for every storage network configured. It mainly obtains configuration parameters through command line prompts for user input, and logs errors and information to a system log.

7.0.118 Technical description

- **Name**: cluster_storage_init_network
- **Description**: Initializes and configures the settings of the storage network for a cluster.
- **Globals**: cluster_domain (Contains the domain name of the cluster)
- **Arguments**: None.
- **Outputs**: Logs of errors and informational messages regarding the achieved or failed configuration of storage networks.
- **Returns**:
  0 - If the function completes successfully
  1 - If an error occurs (such as missing cluster domain or invalid subnet base).
- **Example Usage**:

```
cluster_storage_init_network
```

7.0.119 Quality and security recommendations

1. Avoid relying on a prompt-based system for configuration. This is prone to human error. Instead, use configuration files or automation scripts.
2. Validate all command-line arguments and network details to prevent malformed inputs.
3. It's good practice to separate the functions for fetching, verifying and setting each network parameter for better modularization.
4. It's important to log all errors, warnings, and information for troubleshooting and debugging purposes. However, make sure sensitive information (like IP addresses or domain names) isn't exposed in a readable format.
5. Implement appropriate error handling to ensure the function fails gracefully when problems occur.
6. Consider using a more robust system for managing global variables instead of bash local variables to increase data safety and decrease misuse.

7.0.120 `_collect_cluster_dirs`

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

Function signature: 21a54dd1fed816cbbecd01967b98b68192e6453200d8c224a543b3e6b074b633

7.0.121 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.

7.0.122 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
```

7.0.123 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.

7.0.124 `commit_changes`

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

Function signature: 511b6937ea1ef6712b99a5da762680ed0c87362cbcc9d802fcf9ddecb476ae51

7.0.125 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.

7.0.126 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
```

7.0.127 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.

7.0.128 `config_get_value`

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

Function signature: 3770ff8ebc08ace029f2496144d50418a387d3129f3a483c96ff3865b2d9962d

7.0.129 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 configura-

tion value is found, the function returns a default value.

7.0.130 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"`

7.0.131 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.

7.0.132 `configure_kickstart`

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

Function signature: b9c974579a4cf0918b1f523ab5546c0fabf4f4fe0d6a0a4ab77a23765ef1cbca

7.0.133 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.

7.0.134 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`

7.0.135 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.

7.0.136 `count_clusters`

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

Function signature: 35e810345753544ce73618109f16ad97d33b5185c8ff2d374a8837aa569e5960

7.0.137 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.

7.0.138 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
```

7.0.139 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.

7.0.140 `create_config_dnsmasq`

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

Function signature: e8e5c3ae68c6edbb046d7d24d90b6f00e97a7553b5f4271d29d93e11655d696a

7.0.141 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.

7.0.142 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`

7.0.143 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.

7.0.144 `create_config_nginx`

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

Function signature: d3d22d399af4c7c80fb8449bf421c864014acf615622669621bd6f5a2999ef5e

7.0.145 Function overview

The `create_config_nginx` shell function is designed to create and set up a configuration file for the nginx server. It begins by sourcing the active cluster file (if it exists) and defining the path for nginx's configuration file. The function then uses a heredoc (`<<EOF`) to write the server configurations, including worker processes, user and events into the nginx configuration file.

7.0.146 Technical description

Name: `create_config_nginx`

Description: This function creates an nginx configuration file with appropriate server settings.

Globals: There is one global variable that this function interacts with: - `HPS_SERVICE_CONFIG_DIR`: The directory in which the nginx configuration file is located.

Arguments: This function accepts no arguments.

Outputs: This function outputs an `info` level log message about the configuration of nginx server.

Returns: The function doesn't return any explicit value, since its primary task is writing to a file. If this operation is successful, it will implicitly return `0`. Else, it will return whatever error code is thrown by the failing command inside the function.

Example usage:

```
create_config_nginx
```

7.0.147 Quality and security recommendations

1. Implement a feature to validate the nginx configuration file after it's been written. The `nginx -t` command could be used for this.
2. Consider using strict mode (`set -euo pipefail`) to handle potential errors.
3. Add descriptive comments to the script for easier understanding and debugging.
4. Configure the function to accept inputs such as worker_processes, user, worker_connections as arguments so it can be used more flexibly.
5. Use ShellCheck (or a similar linter tool) to analyze the script for potential issues related to robustness, portability, and maintainability.
6. Implement error handling. For example, check whether the `HPS_SERVICE_CONFIG_DIR` directory exists before trying to write to it.
7. Consider encrypting sensitive information that might appear in logs or configuration files.

7.0.148 depend

Contained in `lib/host-scripts.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: 89406247984185d00547bde8e948365eab651f57a6d007cee8af08dc83a26713

7.0.149 Function Overview

The `depend()` function is a built-in script function used for managing service dependencies in Gentoo's OpenRC init system. This function sets the dependencies that need to be met for a service to start. In the given example, the service needs the `net` service, uses the `docker`, `libvirtd`, `libvirt-guests`, `blk-availability`, `drbd` services, and must start after the `time-sync` service.

7.0.150 Technical Description

Name: `depend` **Description**: The `depend` function is used for setting up the dependencies for the services. It's an important helper function in the OpenRC init system.

Globals: None

Arguments: No explicit arguments are defined for the function. However, functions like `need`, `use`, and `after` use positional parameters to define the dependencies.

Outputs: This function does not produce any user-level output.

Returns: It doesn't return any value. The function calls to `need`, `use`, and `after` functions work purely by side-effect.

Example usage:

```
depend() {
    need net
    use docker libvirtd libvirt-guests blk-availability drbd
    after time-sync
}
```

7.0.151 Quality and Security Recommendations

1. Ensure that you add all the services your script depends on for correct and optimal operation.
2. Avoid circular dependencies as they can cause the system to hang or fail.
3. It is essential to be aware that services may start in parallel if possible. Therefore, ensure your service can handle such a scenario.
4. Validate and sanitize any user input that could be passed as arguments, although this function does not directly accept user input.
5. Verify if all the required services for starting a service are legitimate and secure, limiting the possibility of malicious activities.

7.0.152 `detect_call_context`

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

Function signature: f167df149452fd670d9f40bd87d52442f2f5d5153026bdfcab5f9c60997d7f96

7.0.153 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".

7.0.154 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.

7.0.155 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.

7.0.156 `detect_client_type`

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

Function signature: a5996dd77379049ae4564d5c7eaab09a5189d239c610eea12c0d452cd96097e7

7.0.157 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".

7.0.158 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
```

7.0.159 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.

7.0.160 `detect_storage_devices`

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

Function signature: 26407abf67962b945819ec70c2d91c7a26b60c144eeb209e22298b013ba307ed

7.0.161 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.

7.0.162 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.

7.0.163 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.

7.0.164 `disks_free_list_simple`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: 23bd3ac7877a7c90a934c9f11fb7a056bf39f675410585baaaa84e107157944c

7.0.165 Function overview

The function `disks_free_list_simple()` is a Bash script that lists all available block storage devices in a system. It singles out regular, non-removable hard drives, discounting specific types of block devices such as loop, ram, md, dm devices, or those with partitions. Moreover, the function ignores devices that are mounted, belong to LVM2, Linux RAID, ZFS storage pool, or exist in a zpool. Whenever possible, it prefers World Wide Name (WWN) identifier for device representation.

7.0.166 Technical description

Below is a technical block describing the function:

- Name: `disks_free_list_simple()`
- Description: This function scans and lists the available block storage devices, taking into consideration specific exclusions.
- Globals: [None]
- Arguments: [No arguments are used with this function]
- Outputs: The function will output a list of free storage devices based on the criteria and filters it applies.
- Returns: No value returned.
- Example usage: After defining the function in bash, simply call `disks_free_list_simple` to use. Note this should be used in a Bash environment such as a script or the command line.

7.0.167 Quality and security recommendations

1. Ensure that you have proper permissions before running this function. It requires access to low-level disk information which might be restricted.
2. Extend error-handling mechanisms to include a fallback or feedback in case the function is not able to execute the commands correctly.
3. Validate that the output of each command is as expected, dealing particularly with edge cases where there might be unique devices or mounting schemes at play.
4. Carefully consider the security implications of exposing low-level hardware information. Apply necessary restrictions when and where needed, such as on a multi-user system.
5. Implement testing methodologies to maintain the function's integrity and efficiency.

7.0.168 `download_alpine_release`

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

Function signature: f1438a9265c0b60b2947c704f30ee5980245150741b8578a6cb28e185b7cd407

7.0.169 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.

7.0.170 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
```

7.0.171 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.

7.0.172 `download_file`

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

Function signature: 77275d6db2a730d2f09e1f7111242d9607b215be97269a81302557f4e047a820

7.0.173 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.

7.0.174 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"
```

7.0.175 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.

7.0.176 `download_iso`

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

Function signature: a769efc53df917e64e3dbdfb8acb70dff4b4cb4a89efd3b55a689c47cde86e91

7.0.177 Function overview

The function `download_iso` retrieves a specific version of an Operating System (OS) installation ISO file based on certain parameters such as the CPU architecture, the manufacturer, the name and version of the OS.

7.0.178 Technical description

The function is defined as follows:

- **Name:** `download_iso`
- **Description:** Downloads an ISO file for a certain OS from a base URL, builds the proper download URL and the filename of the downloaded ISO file, checks if the ISO file already exists, and if not, downloads the ISO file and saves it to a designated directory. In case of download failure, it cleans up the failed download file. Currently, only “rockylinux” is supported as the OS name.
- **Globals:** No global variables used.
- **Arguments:**
  - `$1: CPU architecture`
  - `$2: Manufacturer`
  - `$3: OS name`
  - `$4: OS version`
- **Outputs:** Prints status messages about the download process, such as whether the ISO file already exists or if the OS variant is unsupported, the status of ISO download process, etc.
- **Returns:** `0` if the ISO file already exists or if it was successfully downloaded; `1` in case of any failures, such as unsupported OS variant or a failed download.
- **Example usage:** `download_iso x86_64 intel rockylinux 10`

7.0.179 Quality and security recommendations

1. Consider adding more OS support, not just Rocky Linux.
2. Catch and properly handle potential problems, such as issues with the directory creation (`mkdir -p "$iso_dir"`), to prevent unexpected results or vulnerabilities.
3. Implement checksum validation after download to ensure the downloaded ISO file is correct and wasn’t tampered with during transit.
4. Refactor the case block for OS names into separate functions for better readability and maintainability.
5. Use secure coding practices and constant code reviews to avoid potential security issues.
6. Test the function thoroughly under different conditions and scenarios to ensure it behaves properly.

7.0.180 `export_dynamic_paths`

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

Function signature: a6b2a47e08ed460524c491151fd944d515572f0fe9d26a1a2d5d310ad72b99b5

7.0.181 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.

7.0.182 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'`

7.0.183 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.

7.0.184 `extract_iso_for_pxe`

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

Function signature: 291539ccd925935805dd8c7d2a52eeb587e5f13f95dab12edd40bdbe8cf2a210

7.0.185 Function overview

This function `extract_iso_for_pxe` is primarily used to extract ISO files to a specific directory for PXE (Preboot eXecution Environment). The function is provided with the architecture, manufacturer, operating system details and version, and it tries to extract the corresponding ISO file if it hasn't been extracted before. If anything goes wrong during the extraction process, an error message is printed and the function returns with a failure code.

7.0.186 Technical description

- **name**: `extract_iso_for_pxe`
- **description**: Extracts an ISO file under a specified directory for PXE booting.
- **globals**: [`HPS_DISTROS_DIR`: Directory where the ISO file is to be extracted]
- **arguments**: [`$1`: CPU architecture, `$2`: Manufacturer, `$3`: Operating system name, `$4`: Operating system version]
- **outputs**: Prints status messages on console, indicating the progress of the operation.
- **returns**: Returns status code `0` if the operation is successful, or `1` if otherwise.
- **example usage**:

```
extract_iso_for_pxe "x86_64" "dell" "ubuntu" "20.04"
```

7.0.187 Quality and security recommendations

1. Ensure that the function parameters, especially `iso_dir` and `extract_dir` are properly sanitized to avoid directory traversal attacks.
2. Implement checks for the existence of the `bsdtar` utility before proceeding with the extraction. If `bsdtar` is not found, exit the function early with a suitable return code and message.
3. Perform more robust error handling. For example, it would be helpful to check whether the directory creation was successful before proceeding with the extraction.
4. Consider adding additional checks or accept only whitelisted inputs to reduce possibilities of misuse or error.
5. Input validation can go beyond the file system. For instance, validate the architecture and OS version parameters to confirm they're valid before proceeding.

7.0.188 `extract_rocky_iso_for_pxe`

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

Function signature: 52e8a2e1e3a65096a2e0e1ac3696f4c21521a7cc3c3b006a7056ede3446b2ada

7.0.189 Function overview

The function `extract_rocky_iso_for_pxe()` is created to extract a Rocky Linux ISO for PXE (Preboot Execution Environment). This extraction is required to prepare the PXE boot environment. The function takes in three arguments: the path to the ISO, the version of the Linux OS, and the CPU architecture information. It then prepares a location to extract the ISO contents and performs the operation using bsdtar or fuseiso. If neither utility is available, the function logs an error message and exits with a non-zero status code.

7.0.190 Technical description

- **name**: extract_rocky_iso_for_pxe
- **description**: The function aims to extract a Rocky Linux ISO for PXE (Preboot Execution Environment).
- **globals**:
  - VAR: `HPS_DISTROS_DIR`: a variable holding the base directory for Linux OS distributions.
- **arguments**:
  - $1: `iso_path`: Path to the Rocky Linux ISO file.
  - $2: `version`: The version number of the Linux OS.
  - $3: CPU: The required CPU architecture.
- **outputs**:
  - Logs about the extraction process.
  - An error message if neither `bsdtar` or `fuseiso` commands are installed.
- **returns**:
  - 0 if the Rocky Linux ISO extraction is successful.
  - 1 if neither `bsdtar` or `fuseiso` commands are installed.
- **example usage**:

```
extract_rocky_iso_for_pxe "/path/to/iso" "8.4" "x86_64"
```

7.0.191 Quality and security recommendations

1. Always sanitize inputs to the function to prevent any possible malicious or unintended actions.
2. Make sure to check the existence of the ISO file passed as an input, and fail fast, if it doesn't exist.

3. Use full paths to commands like bsdtar, fuseiso, etc. to avoid dependency on PATH and prevent potential "command not found" errors or command injection vulnerabilities.
4. Be mindful of returning appropriate status codes on failure scenarios, it helps to debug faster.
5. Consider providing more informative and meaningful log messages for clarity about each operation.
6. Always handle potential errors in filesystem operations like mkdir, cp, etc.
7. In the "globals" section, important potential side effects of using global variables in a function like potential conflicts with other scripts/environment variables should be kept in mind.

7.0.192 `fetch_and_register_source_file`

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

Function signature: 97650ca173393457a3e6151b2fbb0a997e425c3010fc3c13018364e050618d70

7.0.193 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.

7.0.194 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"
```

7.0.195 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.

7.0.196 `fetch_source_file`

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

Function signature: 9851dec43ce9f3e94856800874741f8ff28deda9346709daaa88282763e49999

7.0.197 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.

7.0.198 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"
```

7.0.199 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.

7.0.200 `find_hps_config`

Contained in `lib/functions.sh`

Function signature: b3ba10a8967a3088d5d8dae7f4bd970972f7563b406d9440431b21d23e4b538d

7.0.201 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.

7.0.202 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)`

7.0.203 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.

7.0.204 `format_mac_colons`

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

Function signature: 95ebd5c4198ab3943c5263e6ec4e563812216afc2059562fef0d3c1357dc53ca

7.0.205 Function Overview

The `format_mac_colons` function is designed to validate a MAC address input, convert it to lowercase, and insert colons in the MAC address format. This simplifies the process of formatting MAC addresses to adhere to standard MAC address representation.

7.0.206 Technical Description

Function name: `format_mac_colons`

Description: This function validates a supplied MAC address to ensure it consists of exactly 12 hexadecimal characters. After validation, it converts the MAC address to lowercase and inserts colons in the appropriate position to adhere with the standard representation of a MAC address.

Globals: None

Arguments: - `$1`: This argument should be a MAC address of 12 hexadecimal number.

Outputs: If valid, a MAC address with colons inserted would be produced. If invalid, the function will output an error message directed to the standard error output (stderr).

Returns: - `1`: if supplied MAC address is invalid. - Formatted MAC address: if the supplied MAC address is valid.

Example usage:

```
format_mac_colons "123456abcdef"
```

7.0.207 Quality and Security Recommendations

1. Input Sanitization: Ensure that the input to the function is properly sanitized before it is processed by the function.
2. Error Handling: The function should have plans to handle unexpected inputs (like a null input or an input with non-hexadecimal characters) and not just invalid length.
3. Include a clear and descriptive comment on what the function does at the beginning of the function. This includes listing out all assumptions, preconditions and postconditions.
4. Add more detailed output messages that can guide the user on the type and format of input the function requires whenever invalid inputs are provided.
5. It would be beneficial to provide more return codes to cover other potential error situations, such as an empty input string. Making return codes more descriptive can make debugging simpler.
6. Always ensure to follow the least privilege principle - only give the minimum amount of permissions necessary for the function to execute.

7.0.208 `generate_dhcp_range_simple`

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

Function signature: d040dcc216635e3158248e2fd6c7c0ba131ed32bec1b3b2c1708b14f4efd6311

7.0.209 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.

7.0.210 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"`

7.0.211 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.

7.0.212 `generate_ks`

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

Function signature: b973477b6d7653d80151f03150fadcfc0d8f1f85a77f353025a267d5257c8529

7.0.213 Function overview

The function `generate_ks()` handles the generation of a kickstart file, used to automate the installation process of an operating system. This function exports different variables containing configuration parameters needed for the OS installation and finally prepares the installation script for rendering.

7.0.214 Technical description

- **Name:** `generate_ks()`
- **Description:** This function accepts two parameters. It exports different variables like macid, HOST_TYPE, HOST_NAME, etc., used for host configuration. It uses helper functions like `hps_log()`, `cgi_header_plain()`, `host_config()`,

`cluster_config()`, and `script_render_template()` to log information, get and set configurations, and render the installation script respectively.

- **Globals:** [macid: first argument, represents MAC ID / Unique Host ID of a host machine, HOST_TYPE: second argument, represents the type of the host machine]
- **Arguments:** [$1: MAC ID / Unique Host ID of a host machine, $2: Type of the host machine]
- **Outputs:** Logs about the function calls and configurations, Along with the final render of the installation script.
- **Returns:** Does not return any value. However, the function could be interrupted and denote failure with a return of 1.
- **Example Usage:**

```
generate_ks "00:0c:29:c0:94:bf" "CentOS"
```

7.0.215 Quality and security recommendations

1. Avoid usage of global variables as much as possible to reduce side effects and improve function modularity.
2. Input validation and error handling should be more rigorous. Check for the validity of MAC addresses before using them.
3. Preferably restrict access to root or privileged users to reduce potential security risks.
4. Avoid inlining large scripts. Use a separate file to manage large scripts.
5. Implement a logging mechanism to record errors and important events during installation.
6. If possible, encrypt sensitive data like IP addresses, hostnames, and other network details.

7.0.216 `get_active_cluster_dir`

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

Function signature: bea86c9b08f47ce60a2602b8a8391990588b8ca0b96c0252ba0070cc3623cfce

7.0.217 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.

7.0.218 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
  ```

7.0.219 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.

7.0.220 `get_active_cluster_file`

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

Function signature: 6461e5f5814cbc510b1bad68822a5b0d8eb3c58618d50b5418200c62ea6dff01

7.0.221 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.

7.0.222 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
```

7.0.223 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.

7.0.224 `get_active_cluster_filename`

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

Function signature: 20bf828d972eed107690831358c6e9148058e88842050f23b8944d669bfab00a

7.0.225 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.

7.0.226 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)
```

7.0.227 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.

7.0.228 `get_active_cluster_hosts_dir`

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

Function signature: 5e8124c7c913768226948ae7120083b2568d70860f21c6599d89e81df548ee7d

7.0.229 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.

7.0.230 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
```

7.0.231 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.

7.0.232 `get_active_cluster_info`

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

Function signature: 51d7c68169c79cb4271c76d4914a13afce322dd3dab4f93ccfeba936798070ea

7.0.233 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.

7.0.234 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`

7.0.235 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.

7.0.236 `get_active_cluster_link`

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

Function signature: 8840658f2d249224284adc89b84feddc787683b794880b06d7f3d566dc412130

7.0.237 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.

7.0.238 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
  ```

7.0.239 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.

7.0.240 get_active_cluster_link_path

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

Function signature: 00422606e36c805412e226764ea91ac8d9620442c34f9c3bf83c8de949497756

7.0.241 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.

7.0.242 2. Technical Description

7.0.242.1 Name

get_active_cluster_link_path

7.0.242.2 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.

7.0.242.3 Globals

[HPS_CLUSTER_CONFIG_BASE_DIR: The directory containing the cluster config files]

7.0.242.4 Arguments

This function does not require any arguments.

7.0.242.5 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

7.0.242.6 Returns

This function will return 0 on successful execution.

7.0.242.7 Example Usage

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

7.0.243 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.

7.0.244 `get_active_cluster_name`

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

Function signature: 56eb8b1d52908fb62f61b716d0ffccedd95bf6c229314f23d9a10385163e0cfd

7.0.245 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.

7.0.246 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"
```

7.0.247 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.

7.0.248 `get_all_block_devices`

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

Function signature: add7deb6a087d72238984be839fb5488e13aff3ae7251f93f2c1814d79162625

7.0.249 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.

7.0.250 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.

7.0.251 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.

7.0.252 `get_client_mac`

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

Function signature: 8cf7b608ec665ff47f16996d542d5d3ed0c9392116e453e8abfcc00eab616973

7.0.253 Function Overview

The function `get_client_mac` is used to extract the MAC address corresponding to a given IP address in a local network. It sends a ping to trigger an ARP update, which is then parsed for the required MAC address. If the initial method does not succeed, it uses the `arp` command as a fallback and returns a normalized MAC address.

7.0.254 Technical Description

- **Name:** get_client_mac
- **Description:** This function uses IP address to get its corresponding MAC address from the Address Resolution Protocol (ARP) cache.
- **Globals:** None

- **Arguments:**
  - `$1`: The IP address of the client.
- **Outputs:** Normalized MAC address related to the IP address if found.
- **Returns:**
  - 1 if the IP address is not valid or MAC address is not found.
  - MAC address corresponding to given IP address in normalized form.
- **Example Usage:**
  - `get_client_mac 192.168.1.1`
  - `MAC_ADDRESS=$(get_client_mac 192.168.1.1)`

7.0.255 Quality and Security Recommendations

1. Ensure the user has required permissions to run the `ping`, `ip neigh` and `arp` commands to avoid permission denied errors.
2. Include error handling for cases where ARP does not contain an entry for the given IP address, emitting appropriate error messages.
3. Consider including a validation check for the normalized MAC address before returning it.
4. Use secure command execution to prevent injection attacks.
5. Consider returning a standardized error value, such as a null address or a specific error string, if the MAC cannot be found.

7.0.256 `get_cluster_conf_file`

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

Function signature: fc7feb39024383e4cb59d1bec6341e1f7248d7dd978aa33bac047e372cc4613e

7.0.257 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.

7.0.258 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
```

7.0.259 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.

7.0.260 `get_cluster_dir`

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

Function signature: f5e9f656e463f433ab0e3bfed0da73d9166cc4e5802421827e55bef022685a0a

7.0.261 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.

7.0.262 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
```

7.0.263 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.

7.0.264 `get_cluster_host_hostnames`

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

Function signature: 0a454d05c4d3133d2a88b66d07816dd02a6a5ae702c43b33ecd266836eee2f6a

7.0.265 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.

7.0.266 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"`

7.0.267 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.

7.0.268 `get_cluster_host_ips`

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

Function signature: 57b1cadd8f6aaacb69bc449fe7bdb26cd5e773b01b8fcd81a1c730f7042e63c0

7.0.269 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.

7.0.270 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
```

7.0.271 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.

7.0.272 `get_device_bus_type`

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

Function signature: 71e77c52eb2ba1d481c9ef51c928878b08e2c1088eeb0bf2fbe433c62633a476

7.0.273 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".

7.0.274 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`

7.0.275 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.

7.0.276 `get_device_model`

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

Function signature: ec59456e4d8546a811f0f4533daf51da08474fb4ada4f1efb264e30d2fea7091

7.0.277 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".

7.0.278 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
```

7.0.279 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.

7.0.280 `get_device_rotational`

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

Function signature: 1002aec4163a82a48171eea735ad8700633333695a4b75dcb9ecc3317d14222f

7.0.281 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"`.

7.0.282 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.

7.0.283 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.

7.0.284 `get_device_serial`

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

Function signature: 8e1b210627973a0cc629e646a16460819b61ae08954c5dba162358f2b6cb4532

7.0.285 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.

7.0.286 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`

7.0.287 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.

7.0.288 `get_device_size`

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

Function signature: 1d4e207d13b80b5424975cd10a3ed7197f78e6832fbbf6662e417abdd6def6d2

7.0.289 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".

7.0.290 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"`

7.0.291 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.

7.0.292 `get_device_speed`

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

Function signature: a3925bc33db6a5252197ab449a78edcdfb4d9117f6e067cedb4b22c2eaea3a3e

7.0.293 Function overview

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

7.0.294 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"
```

7.0.295 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.

7.0.296 get_device_type

Contained in lib/functions.d/storage_functions.sh

Function signature: c04dccee20f8398f1cc00c1f339464f78c0ea09416025a9627dcbd45477ea68d

7.0.297 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”.

7.0.298 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
```

7.0.299 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.

7.0.300 get_device_usage

Contained in lib/functions.d/storage_functions.sh

Function signature: 859cf1265955125053e13b61c6c4bfa14f3715f652ae3ee914eb7e3927870233

7.0.301 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”.

7.0.302 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
```

7.0.303 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.

7.0.304 `get_device_vendor`

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

Function signature: e635f4842a790d321a2d2bf3371ad26f62f2881ced6effdff7eaea8887f48cf1

7.0.305 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.

7.0.306 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.*

7.0.307 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.

7.0.308 `_get_existing_zpool_name`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: cc41e1bc5b39a7a3e09eef4edc9166c5f5783d182d1cd860936367173015ae84

7.0.309 Function Overview

This function, `_get_existing_zpool_name`, is used for obtaining the existing ZFS pool name from a remote host. If the ZFS pool name is present, the script prints the name to stdout and returns a zero exit status. If the ZFS pool name is not found, the function returns a non-zero exit status.

The code processes options (e.g. `strategy`, `mountpoint`, `force`, `dry-run`, `no-defaults`) and calls helper functions like `zpool_name_generate` and `zfs_get_defaults` for specific functionalities. It also checks the ZPOOL_NAME before proceeding, validates the policy rules, generates the pool name and selects a disk based on the strategy.

Subsequently, the script performs the creation of the pool, persists the host variable ZPOOL_NAME and provides appropriate logging and error handling throughout the execution.

7.0.310 Technical Description

- Function name: `_get_existing_zpool_name`
- Description: This function is responsible for obtaining the name of an existing ZFS pool from a remote host.
- Globals:
  - `ZPOOL_NAME`: The name of the ZFS storage pool.
- Arguments:
  - `$1`: Command line arguments and options.
  - `$2`: Value of argument where it applies.
- Output: Prints the ZFS storage pool name to stdout if exists.
- Returns: Exit status `0` when pool name is found, `1` or `2` when there is an error.
- Example Usage: `_get_existing_zpool_name --strategy first --mountpoint /tmp --dry-run`

7.0.311 Quality and Security Recommendations

1. The function could benefit from additional input validation. Checking whether the provided arguments are recognizable before their first usage could prevent unexpected behaviors.
2. The error messages can be improved to be more descriptive. For example, stating precisely why a specific helper function is missing could allow an easier troubleshooting experience.
3. The function relies heavily on other functions (helpers), their availability and their output. As such, this function could be prone to break if any of the helper functions are modified. Robust integration tests could help catch these issues.
4. Security-wise, the script does not seem to sanitize inputs when making system calls. This could potentially lead to command injection vulnerabilities. Performing

rigorous input validation and sanitization throughout the script would help mitigate this risk.
5. Consider implementing a more comprehensive logging functionality, such as logging every action and result, which can help in debugging and can be crucial in live environments.
6. If possible, make the function idempotent. The function should give the same output and have the same side effects, regardless of how many times it's run with the same inputs. This would ensure predictable behavior.

7.0.312 `get_external_package_to_repo`

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

Function signature: a00cf7324f6795dbbcefc882d9a0346242d31cdeaef0ed00e85c79720244ebc9

7.0.313 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.

7.0.314 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"`

7.0.315 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.

7.0.316 `get_host_conf_filename`

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

Function signature: 8b543e00f7c0c713c4d8c537217c6a9c6addf01207ee1dfb10ecf00e0041e3e7

7.0.317 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.

7.0.318 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"
```

7.0.319 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.

7.0.320 `get_host_type_param`

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

Function signature: 2aa086f62cbb99876d8c7312d176dfce1f88af4d006d3f9a64537fa96e3f9008

7.0.321 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.

7.0.322 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"`

7.0.323 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.

7.0.324 `get_interface_network_info`

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

Function signature: 4e173360adff11eb87d4f357c81a5a4827f74ea723adff69c69fd4d1c4dcae15

7.0.325 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`.

7.0.326 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.

7.0.327 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.

7.0.328 `get_keysafe_dir`

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

Function signature: 25a0229bfcf5422d4fe07f68f0d2261ee643e57fcff68bbc63ba33c0f41af39e

7.0.329 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.

7.0.330 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
```

7.0.331 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.

7.0.332 get_latest_alpine_version

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

Function signature: f1404bc114d5e831d9237d9abb99e8b1e61a8b81eef840b822e47765fdfb7591

7.0.333 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.

7.0.334 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

7.0.335 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.

7.0.336 `get_mac_from_conffile`

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

Function signature: e02347fb032c6871b64d63fdf633e3ff78088698ab33bbdb1671faa21c210ea9

7.0.337 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.

7.0.338 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
```

7.0.339 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.

7.0.340 `get_network_interfaces`

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

Function signature: 5f860b3e71cc7a8b5a1076bb49f656f0d7ddd17d0a523e3e4173aba7dffe4dc9

7.0.341 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.

7.0.342 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.

7.0.343 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.

7.0.344 `get_path_cluster_services_dir`

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

Function signature: 614369342a40a8d8eacfb7e30ad0b4a1d719139038d55af309dbc419dcd2cb3b

7.0.345 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.

7.0.346 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
```

7.0.347 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.

7.0.348 `get_path_supervisord_conf`

Contained in `lib/functions.d/configure-supervisor.sh`

Function signature: 5ead4ebed48cedb7a928e3c048d0d0b546746f48cc20520b8e2cdef5bd83bbad

7.0.349 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.

7.0.350 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
```

7.0.351 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.

7.0.352 `__guard_source`

Contained in `lib/functions.sh`

Function signature: e8beb9b32cabb9e73dba64f7b102ad5f1112590b455a914144bd65e5d3001168

7.0.353 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.

7.0.354 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
```

7.0.355 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`.

7.0.356 `handle_menu_item`

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

Function signature: a9ac642f70b3f86fdb4bd88ce97a68bea7e050b651ec8c852b5184fec386e901

7.0.357 Function Overview

The function `handle_menu_item()` is designed to manage various IPXE menu functions for a given machine which is identified by its MAC address. Depending on the provided arguments, the function can initialize the menu, install hosts, recover, display information, configure/unconfigure, reboot, boot locally, reinstall, rescue, install, or force installation. It uses case statements to parse the command and execute different functionality depending on the input it receives.

7.0.358 Technical Description

- **Name**: `handle_menu_item`
- **Description**: This function handles different functions related to IPXE menus, such as initialising, installing, unconfiguring, rebooting, etc. It uses a case statement to identify the function to be performed.
- **Globals**: None.
- **Arguments**: `[ $1: item - the specific ipxe menu command to execute, $2: mac - the MAC address of the machine for which the operations are to be performed ]`

- **Outputs**: Information related to the status of the execution of the IPXE menu item handled by the function.
- **Returns**: The function does not return anything. However, it may change the state of the host configuration.
- **Example Usage**: `bash handle_menu_item "reboot" "00:0a:95:9d:68:16"`

7.0.359 Quality and Security Recommendations

1. Enforce input validation not only for `item` but also for `mac` to ensure the inputs provided are in the correct format, improving reliability and preventing potential injection attacks.
2. Ensure there is a default error or exception handling mechanism in place for cases when an unexpected error occurs during the execution of a command, improving fault tolerance.
3. Include logging for all important actions and decisions made during the function's execution, providing visibility into its operations and aiding in debugging.
4. Centralize error messages to facilitate updates and maintain consistency across the application, improving maintainability and user experience.
5. Enforce least privilege principle by only assigning minimum required permissions to the function (or the user executing the function) to prevent potential misuse and mitigate damage from potential vulnerabilities.

7.0.360 `has_sch_host`

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

Function signature: 8239c1c521d6e5ec4f193188027bf12f8f78909eebda3b607d1d03cc1835a65d

7.0.361 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.

7.0.362 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
```

7.0.363 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.

7.0.364 `host_config_delete`

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

Function signature: 879c4c1c59478ffcfd437a6d710d463fd2811a08a4bc4f57d5dd6d62b5a37709

7.0.365 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.

7.0.366 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"
```

7.0.367 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.

7.0.368 `host_config_exists`

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

Function signature: 34de36858dffe5d99bd23615eca3110b87f3a65c04849bd3f82c9ba9fb2194ea

7.0.369 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.

7.0.370 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
```

7.0.371 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.

7.0.372 `host_config`

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

Function signature: e3395cb667c86565ac27cd12907de2a73dd934fe68e2741bcb6178e42eb80743

7.0.373 Function overview

`host_config` is a Shell function that allows operations on configuration values associated with a host's MAC address. The function is capable of getting a configuration value, checking if a key exists, validating a key's value, and setting a key's value. The configuration data is stored in associative array `HOST_CONFIG`, and loaded from a configuration file located at `${HPS_HOST_CONFIG_DIR}/${mac}.conf`.

7.0.374 Technical description

- **Name:** host_config
- **Description:** A Shell function to get, check and set configuration values for a specific MAC address.
- **Globals:**
  - `__HOST_CONFIG_PARSED`: Indicates if a config file has been parsed.
  - `__HOST_CONFIG_MAC`: The currently loaded MAC address.
  - `__HOST_CONFIG_FILE`: The configuration file to load.
- **Arguments:**
  - `$1`: The MAC address of the host.
  - `$2`: The command to run on the configuration (get, exists, equals, or set).
  - `$3`: The key in the configuration to operate on.
  - `$4`: The value to set in the configuration, if the command is `set`.
- **Outputs:** Logs an error message when an invalid key or command is provided.
- **Returns:** `0` if operation is successful, `1` if a specified key does not exist, `2` if an invalid key format or command is provided, `3` if it fails to create a config directory or to write a configuration file.
- **Example usage:**

```
host_config '00:11:22:33:44:55' 'get' 'my_key'
host_config '00:11:22:33:44:55' 'exists' 'my_key'
host_config '00:11:22:33:44:55' 'equals' 'my_key' 'my_value'
host_config '00:11:22:33:44:55' 'set' 'my_key' 'my_value'
```

7.0.375 Quality and security recommendations

1. Use printf instead of echo for more predictable and consistent output.
2. Be careful with the execution of `host_post_config_hooks` function, make sure it does not introduce security vulnerabilities.
3. Ensure the configuration directory and file have correct file permissions to prevent unauthorized access to the configuration.

7.0.376 `host_config_show`

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

Function signature: 84f65b3dca74b99c5f2890c67f216c67c0deb77fe856d7ec3d6f8447b05622e8

7.0.377 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.

7.0.378 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"`

7.0.379 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.

7.0.380 `host_initialise_config`

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

Function signature: bbb7fcbeb2dfdec2230b65cd3f866dfafa5af028145e799ac74c70b92faabf24

7.0.381 Function overview

The `host_initialise_config` function in Bash is designed to initialise the host configuration for the provided MAC address. It validates if the MAC address is provided and determines the hosts directory for the active cluster. If the hosts directory does not exist, it creates one. And then it sets an initial state using `host_config`. It will fail to set up the initial state it will log an error message and terminate the function returning 1. If all operations succeed, it logs an information message with the config file path and returns 0.

7.0.382 Technical description

- name: `host_initialise_config`
- description: Initialises a host configuration based on a given MAC address. Creates the corresponding directories if they are non-existant, and sets the initial state.
- globals: None
- arguments:
  - `$1`: MAC address that needs the initialisation of a host configuration.
- outputs: Logs various states of configuration initialisation, including errors and informational messages about successfully created directory and initialised host config.
- returns: `0` if the host configuration was successfully initialised and `1` if the MAC address was not provided or any issues occurred during the config initialisation process.
- example usage: `host_initialise_config 00:0a:95:9d:68:16`

7.0.383 Quality and security recommendations

1. The function could include more detailed logging of errors, including the specific error messages that resulted from the function calls. This would allow for faster and more accurate troubleshooting of any issues encountered during runtime.
2. The function could include functionality to validate the format of the provided MAC address before proceeding with the rest of the function. This would prevent potential errors or unexpected behavior if an invalid MAC address is provided.
3. To improve security, consider adjusting the permissions on the created folders and files to limit access to those who need it. Providing unrestricted access might lead to potential security vulnerabilities. Make sure that any information related to errors (such as directory locations) is not displayed to regular users, to prevent potential information leaks.
4. To ensure the function behaves as expected, implement unit testing and include edge cases such as an empty string, a non-existent directory, or a MAC address that

already has a corresponding configuration.

7.0.384 `host_network_configure`

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

Function signature: 8cd2d827ef31d1bf69059fb464c2261c8c3d0c359949b29cc3d9e2124e82ea48

7.0.385 Function overview

The `host_network_configure` function is primarily responsible for assigning a unique IP and hostname to a specific host identified by a MAC address in a cluster network. This assignment is necessary for effective networking within the cluster. Apart from this, the function also conducts the validation of various inputs. It calls helper functions to get: the current configuration using `cluster_config helper`, the host's current IP address using `host_config helper`, assigned IPs and existing hostnames using `get_cluster_host_ips and get_cluster_host_hostnames helpers`. The function relies on other helper functions for data processing including: validating IP addresses, converting CIDR to netmask, and converting IP addresses to integers and vice-versa. The function also handles errors and logs information at every crucial step.

7.0.386 Technical description

- **name**: host_network_configure
- **description**: Assigns a unique IP address and hostname to a specific host in a cluster network using its MAC address as identification. The function validates all the inputs and handles errors effectively.
- **globals**: None
- **arguments**: [$1: MAC address of the host to be configured (`string`), $2: the type of host (`string`)]
- **outputs**: None
- **returns**: 0 if successfully configured host; 1 in various failure cases
- **example usage**: `host_network_configure "00:0a:95:9d:68:16" "physical"`

7.0.387 Quality and security recommendations

1. Handling input parameters should cater for malformed inputs (i.e., non-string types for `macid` and `hosttype`).
2. When dealing with IP and hostname validation, additional layers of security checks could be enforced to prevent injection or scripting attacks.
3. Use of a formal logging system would be beneficial for easier future debugging since current logging lacks levels of severity.

4. Code comments are clear but could be enhanced for better understanding of subprocesses within loops.
5. While the function captures errors during the processing of network configuration, printing of these errors would give users more insight into what went wrong.
6. To deal with possible race conditions during IP and hostname assignment, the introduction of locks or a queue system could help synchronize this crucial step.
7. Error handling should perhaps be updated to handle specific error cases, rather than general errors only. This would make the system more robust and easier to debug.
8. For function return values, using named constants instead of numbers could make the code more readable and maintainable.
9. Finally, this function could be broken down further into smaller subfunctions to make it easier to understand and test.

7.0.388 `host_post_config_hooks`

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

Function signature: f0dad0c738e5b5f97b960e78daed3dbb06ffdffd98737aaa8939313fb7223925

7.0.389 Function Overview

The function `host_post_config_hooks()` takes as arguments a key and a value, and performs actions based on the key provided. Within the function, a hook mapping is defined, associating specific keys with specific functions to be performed. If the key provided as an argument is found in the hook mappings, the associated function is executed; the execution is logged and any output is suppressed. If the function fails, a warning message is logged. Regardless of the execution outcome, the function always returns a success status.

7.0.390 Technical Description

- **Name:** `host_post_config_hooks()`
- **Description:** This function takes in a key-value pair and performs specific actions (hook functions) mapped to the key. Status updates are logged.
- **Globals:** *None defined directly within this function.*
- **Arguments:**
  - `$1`: Key to check in the hook mappings.
  - `$2`: Value paired with the key; used for logging purposes.
- **Outputs:** Logs informational and warning messages based on the handling and execution of the hook function.
- **Returns:** Always returns `0` which signifies success in bash.
- **Example Usage:** `host_post_config_hooks "IP" "192.168.1.1"`

7.0.391 Quality and Security Recommendations

1. Ensure error handling is robust: Currently the function handles hook function failures. This could, however, be made more robust by handling more exceptions and adding more informative logs.
2. Consider limiting the keys and values that can be added: To prevent misuse of the function, consider enforcing restrictions or validity checks on the key-value pairs passed.
3. Encrypt sensitive data: If the key-value pairs contain sensitive data, consider encrypting them to enhance security.
4. Always sanitize input: As a general security concept, any inputs passed into functions should be sanitized properly. A check should be added to handle malicious or unintended key-value inputs.
5. Follow the least privilege principle: Make sure that your script is running with the lowest permissions possible. This reduces the chance of the script making unwanted or harmful changes.

7.0.392 `hps_log`

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

Function signature: 7cb7f11ea9e4a792671d9c09325d32e8a4dd46d495d30350236ad1cc427116bf

7.0.393 Function Overview

This bash function, `hps_log()`, is a logging mechanism in the Bash script dedicated to logging events from a particular system. Through this function, messages with different levels can be logged in a well-structured format including timestamp, identifier and log level into a system log file. The log identifier `ident`, the log file directory `HPS_LOG_DIR`, and the log level are adjusted through global variables and arguments.

7.0.394 Technical Description

name: `hps_log`

description: A bash function for logging events in a system. It logs events into a system log file with the log format of timestamp, identifier, log level, and message.

globals: - `HPS_LOG_IDENT`: Used to set the identifier in the log. If not defined, defaults to 'hps'. - `HPS_LOG_DIR`: Used to set the directory where the log file will be saved. Defaults must be set externally.

arguments: - `$1`: Represents the log level for the event. It's converted to uppercase. - `$*`: Represents the message for the log that captures the system event details.

outputs: Writes a log entry to the `hps-system.log` file located at the log file directory `HPS_LOG_DIR`.

returns: Does not return a value but exits the function after writing the log entry.

example usage:

```
HPS_LOG_DIR="/path/to/log/directory"
hps_log "info" "System operation successful."
```

7.0.395 Quality and Security Recommendations

1. It's crucial to ensure the write permission for HPS_LOG_DIR directory to avoid permission denied error.
2. Sanitize user-supplied input to the function. Any user-supplied input incorporated into the log message should be properly sanitized to prevent code injection attacks and log forgery.
3. Implement log rotation and archival strategy. If the logging function is not properly managed, the log file could grow very large which could create a Denial-of-service (DoS) condition on the file system.
4. Add error handling to the logging function, to gracefully handle any errors occurred during logging, such as file write errors.
5. Consider time synchronization and timezone information since log files often serve as a key piece of evidence during incident response and digital forensics.

7.0.396 hps_origin_tag

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

Function signature: 34e636eb4b7c0c9bf98b49ce8416227a48485bd990954c5b7f74feba9f1c472a

7.0.397 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.

7.0.398 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)`

7.0.399 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.

7.0.400 `hps_services_restart`

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

Function signature: a6205428f2faad6f9af5847c527a10dd5eced1de31f2f36738b99daede46ce67

7.0.401 Function Overview

The `hps_services_restart` function is part of the bash scripting language and is primarily used for restarting services in a High Performance System (HPS). Its tasks are executed in sequence including arranging supervisor services, creating a supervisor services configuration, reloading the supervisor configuration, logging the restarting process, and executing post-start tasks for the services.

7.0.402 Technical Description

- **Name:** `hps_services_restart`
- **Description:** This function is responsible for restarting all of the services in an HPS environment. It undertakes several responsibilities including configuring, creating, and reloading supervisor services, logging the whole restart process, and finally triggering post-start functions.
- **Globals:** No global variables are directly addressed by this function.
- **Arguments:** This function does not require any arguments.

- **Outputs:** The function outputs an info log message which includes the results of the `supervisorctl -c "$(get_path_cluster_services_dir)/supervisord.conf"` command, which restarts all services.
- **Returns:** The function does not explicitly return a value. The outcome of the function depends on the success of restarting all services.
- **Example usage:**

```
hps_services_restart
```

7.0.403 Quality and Security Recommendations

1. Always test this function with non-critical services before applying it to a live cluster to ensure its proper operation.
2. Guarantee that each of the invoked functions (like `configure_supervisor_services`, `create_supervisor_services_config`, `reload_supervisor_config`, `hps_log`, and `hps_services_post_start`) securely manage exceptions, errors and return statuses.
3. Ensure that logging is set at an appropriate verbosity level to provide enough detail for troubleshooting any issues that may arise without overcrowding the log files.
4. Validate user inputs or values loaded from files before utilizing them to prevent potential code injection attacks.
5. Make sure access permissions are correctly set, to prevent unauthorized access.
6. Always keep the HPS environment and the related software components updated, as new updates often bring security patches and enhanced performance.

7.0.404 `hps_services_start`

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

Function signature: 253fab38e2efb85dc8064143db95e2df87e1659424b7942e7fa9e5180ce82c59

7.0.405 Function Overview

The `hps_services_start` function works within a Bash context to set up and start all the cluster services within a given set up. The function involves several steps. Initially, it calls the `configure_supervisor_services` function to set up the services. Subsequently, the `reload_supervisor_config` function is triggered to refresh the configuration setup. Then, the `supervisorctl` command is executed with a configuration file present in the directory path returned by `get_path_cluster_services_dir` function, to start all services. Finally, a `hps_services_post_start` function is called which can be utilized to perform post startup operations.

7.0.406 Technical Description

- **Name**: `hps_services_start`
- **Description**: A function provided to set up and start all services in a cluster configuration.
- **Globals**: None
- **Arguments**: None
- **Outputs**: This function does not explicitly output any values.
- **Returns**: This function does not have any return statement, meaning the shell's exit status is left at the result of the last command executed (which in this case is `hps_services_post_start`).
- **Example Usage**: This function is typically run within an environment that has the necessary variables and dependencies, and is simply called as `hps_services_start`.

7.0.407 Quality and Security Recommendations

1. Implement error handling: To enhance the function's reliability, error handling should be incorporated to manage the failures of the intermediate commands.
2. Insert debugging code: As a part of quality enhancements, it would be beneficial to include debugging code which could provide detailed insights into the function's operation when required.
3. Security improvements: If the function is used in a high security environment, it would be important to ensure that the permissions for the supervisor configuration files and the directories involved are restricted.
4. Check for dependency issues: Effort should be made to test the function in an environment similar to the one in which it is supposed to run, to check for any unmet dependencies.

7.0.408 `hps_services_stop`

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

Function signature: 8389cadf05a55bb98c0cc0b40a84671a6cc4e156fdaaafee6f327e84dff29f00

7.0.409 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.

7.0.410 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)

7.0.411 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.

7.0.412 `initialise_cluster`

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

Function signature: d5981ec28473618b5cd51e9c3ade1ff3471cc55b1c1da368f886f31feec27ee4

7.0.413 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.

7.0.414 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"`

7.0.415 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.

7.0.416 `install_opensvc_foreground_wrapper`

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

Function signature: 044d09f5864508cf501804626eabf1e2282f7b5c3a9c290f490b470dcca0b251

7.0.417 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.

7.0.418 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`

7.0.419 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.

7.0.420 `int_to_ip`

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

Function signature: 14a6ccb9401350f58647bcc5d3c386b17ebd884462480f7e5ed07df11d092d11

7.0.421 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.

7.0.422 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.

7.0.423 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.

7.0.424 ips_allocate_storage_ip

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

Function signature: a6398c88aec77a00b6f34730128b56c2ba1aea6dbada64e3b2e36f5054aea1b7

7.0.425 Function overview

The ips_allocate_storage_ip() function in Bash is designed to allocate an IP address from the storage network to a specific Source MAC address (source_mac). It first checks if an IP address has been already allocated to the given MAC, and in that case, returns the existing allocation. If not, it gets the storage network configuration, extracts the network prefix and builds a list of used IPs for this storage network. It then assigns an IP that falls in the range of 100 to 250 and isn't already in use. If it runs out of IPs in the range, it logs an error message and ends the function. Otherwise, the function coordinates the allocation and stores it, logging an information message and returning the new configuration.

7.0.426 Technical description

Name: ips_allocate_storage_ip()

Description: Allocates an IP address from a storage network to a given MAC address.

Globals:

- storage_index: Index of the storage space.
- source_mac: MAC address passed by the n_ips_command framework.

Arguments:

- $1: The index of the storage (default value is 0).
- $2: The MAC address to which an IP address will be allocated.

Outputs: A string formatted "vlan_id:ip_address:netmask:gateway:mtu"

Returns:

- 0 if allocation is successful.
- 1 if no available IPs are found within a specified range in the storage network.

Example usage: ips_allocate_storage_ip 1 "00:11:22:33:44:55"

7.0.427 Quality and security recommendations

1. Consider implementing input validation to ensure that the input MAC address (and the optional storage index) is in valid format.
2. Ensure the file permissions for script are properly secured to prevent unauthorized access or modification.
3. Handle exceptions and errors properly: In this script, if an available IP is not found in the given range, the script simply logs an error message and returns. It could be more useful to have a fallback strategy or retry mechanism here.
4. Keep the script up-to-date with current best practices relating to IP address allocation and network configuration.
5. Consider refactoring some parts of the function to minimize complexity and improve readability and maintainability. You might break up larger sections into smaller helper functions, each with a single, clear responsibility. This will also aid in unit testing individual components of the functionality.

7.0.428 `ip_to_int`

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

Function signature: e1516fc19276d7592ec5f66f1d05a17ea74eeb43a3c726a9bb01fa86aa2a0b9b

7.0.429 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.

7.0.430 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"`

7.0.431 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.

7.0.432 `ipxe_boot_alpine_tch`

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

Function signature: 0cbf6ac35e7e74554ebe07ac18dc511062766cfc76188035270af89de4c88687

7.0.433 Function overview

The `ipxe_boot_alpine_tch` function boots an Alpine Linux system using Internet Packet eXchange Environment (iPXE) with specific settings. The function retrieves the latest version of Alpine Linux, configures the network settings using the host and cluster configurations, and generates an Alpine apkvol file if it doesn't already exist. It then prepares the kernel arguments for the boot process, sets up the iPXE header, and creates the boot script with the necessary kernel and initrd URLs, along with the generated kernel arguments.

7.0.434 Technical description

- **name**: `ipxe_boot_alpine_tch`
- **description**: This function boots an Alpine Linux system using the Internet Packet eXchange Environment (iPXE).
- **globals**: [`HPS_DISTROS_DIR`: Directory where distribution files like apkvol file are kept]

- **arguments**: [`$mac`: mac address of the host to be booted with Alpine Linux, used to retrieve specific host configurations]
- **outputs**: The function outputs a boot script for iPXE to the stdout.
- **returns**: None. The function does not explicitly return a value.
- **example usage**: `ipxe_boot_alpine_tch "$mac"`

7.0.435 Quality and security recommendations

1. Always validate input parameters. In this function, mac address is an input parameter. Ensure that it is a valid mac address before processing.
2. Avoid using hard-coded IP addresses or hostnames. If possible, these parameters should be configurable or defined as constants at the top of your script.
3. Avoid the command injection vulnerability by always escaping or validating any input that is incorporated into shell commands.
4. Error handling needs to be in place wherein each command's return status should be checked to ensure the task was completed successfully before proceeding.

7.0.436 `ipxe_boot_from_disk`

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

Function signature: f26dc79cd8c9ea270e2758702b63d4a607fd3fd40737c5d947de898af37ff1fe

7.0.437 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.

7.0.438 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`

7.0.439 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.

7.0.440 `ipxe_boot_installer`

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

Function signature: c73d78c14795c1c7103969e913c3a8552641445fa9ab836f6ca2f3323647ace0

7.0.441 Function Overview

The `ipxe_boot_installer()` function primarily configures a host for the new network boot installation. It takes two positional arguments, `host_type` and `profile`. If a profile is provided, it sets the `HOST_PROFILE`. If the host type is "TCH", it configures the host for network boot and forces a reboot. Else, it gathers host type parameters, checks if it's already installed, and finally configures the distro path and URL. It then appropriately prepares the distro for PXE Boot based on the OS Name (currently validated for rockylinux) and sets the state as installing.

7.0.442 Technical Description

- **function name:** `ipxe_boot_installer()`
- **description:** Configures a host for network boot installation. Supports rockylinux based distros.
- **globals:**
  - `CPU`: The type of CPU of the host
  - `MFR`: The manufacturer
  - `OSNAME`: The name of the operating system
  - `OSVER`: The version of the operating system
- **arguments:**
  - `$1 (host_type)`: The type of host.

  - `$2 (profile)`: A profile for the host. Optional.
- **outputs:** Debug/Info log messages, and/or installation configurations. In case of errors, outputs back to invoking client.
- **returns:** Nothing directly. It can exit the script in case of boots or failures.
- **example usage:** `ipxe_boot_installer TCH default`

7.0.443 Quality and Security Recommendations

1. Headers should include error handling with proper message propagation; this helps in operations debugging and error handling.

2. Key dependencies, like external scripts/functions used, should be documented.
3. The function does not currently support Debian distros. Future enhancements should cover more distributions for broader usefulness.
4. Use of local variables can be increased for safer namespace and avoid potential global variable overwrites.
5. Where possible, function arguments should be sanity checked or validated.
6. For security purposes, it is strongly recommended to verify the integrity of the distro before execution.

7.0.444 `ipxe_cgi_fail`

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

Function signature: 98eb961ae3ed7dfffc7f4ae68cd1c88aa5f47672ee53b71b29f0428922e8efaa

7.0.445 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.

7.0.446 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"`

7.0.447 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.

7.0.448 `ipxe_configure_main_menu`

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

Function signature: 1f7c5adf70ab3b0d8ce1dc04122eff4ae6c8141a95956ef26ee00221876e111c

7.0.449 Function Overview

The function ipxe_configure_main_menu is used for displaying the main menu options when a host is not configured in a cluster. The function logs and delivers a configuration menu. The menu contains options for a host to install, view configuration, recover from Disaster Recovery Host (DRH), enter rescue shell, boot from local disk, reboot, and set advanced options. Another significant feature is Forced installation which is set by checking global 'FORCE_INSTALL' and can be enabled or disabled from the menu.

7.0.450 Technical Description

- **Name:** ipxe_configure_main_menu
- **Description:** This bash function delivers a configuration menu with several options to manage a host that is not configured within a cluster. The options include various actions such as installing, viewing configurations, enabling forced installations, etc.
- **Globals:** FORCE_INSTALL: An option that forces the installation process on next boot if set to "YES".

- **Arguments:** None
- **Outputs:** Outputs the configuration menu to the terminal
- **Returns:** Doesn't return a value, executes commands based on the user's choice in the menu
- **Example usage:** This function does not require any arguments. An example of usage would be simply calling the function as `ipxe_configure_main_menu`.

7.0.451 Quality and Security Recommendations

1. Ensure that the host config and all command line arguments are properly sanitized to prevent command injection vulnerabilities.
2. It would be useful to check the statuses of operations like 'host_config' and short circuit the execution of the function, in the event of an error.
3. Ensure that the logging mechanism in 'hps_log' properly sanitizes and escapes any string input to prevent log injection attacks.
4. As a quality improvement, among the menu items, 'Recover from Disaster Recovery Host (DRH)' is mentioned as 'NOT YET IMPLEMENTED'. Consider implementing this feature or removing it from the menu to avoid user confusion.
5. For enhancing security while fetching the log, handle the failure case with more than just an echo statement. Prompt the user, or retry fetch.

7.0.452 `ipxe_header`

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

Function signature: f724cbc120f1a1eae5cf985238d6bc44a932f4521832fa8959c47118c5068ed5

7.0.453 Function Overview

The `ipxe_header()` function transmits a pxe header to prevent boot failures. It establishes a series of variables for use in IPXE scripts, then outputs these to the console.

7.0.454 Technical Description

- **Name:** `ipxe_header`
- **Description:** The function initially transmits a pxe header to prevent any boot failures. It sets a couple of global variables (CGI_URL, TITLE_PREFIX) for use in IPXE scripts. The function constructs and prints a specific message structure.
- **Globals:** [CGI_URL: The URL to the boot manager script running on the selected server, TITLE_PREFIX: The title prefix combining the cluster name, mac address, and network IP of the server]
- **Arguments:** [None]

- **Outputs:** Console output which includes a log message, specifics about the server, and information about the client.
- **Returns:** No explicit return value.
- **Example usage:** It's frequently used in the context of IPXE scripting and will typically be invoked without arguments, like so:

  ```
  ipxe_header
  ```

7.0.455 Quality and Security Recommendations

1. It is crucial to double-check all inputs, specifically while fetching images over the network from a specified URL. This safeguard prevents possible vulnerabilities associated with potential malicious content.
2. All user data or potentially sensitive information output to the console should ideally be sanitized or selectively displayed, as it may expose critical information to malicious actors or risk leaking sensitive data.
3. It may be useful to add error handling to address potential failures that may occur throughout the execution of the function. For instance, if `cgi_header_plain` or `imgfetch` fail, there should be appropriate error messages and failure handling.
4. Increase the readability of the code by adding further comments regarding the functionality and working of different code blocks within the function.

7.0.456 `ipxe_host_install_menu`

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

Function signature: 918ca2eb38d0647620466dd5e4bad4af4149bf195f8ce918ee9dd7b4e79751b5

7.0.457 Function overview

The function `ipxe_host_install_menu` is designed to provide a tool for network booting, offering several options for configuration. These options range from the installation of thin compute hosts, storage cluster hosts, to disaster recovery hosts, with or without specific profiles. Users have the freedom to choose the profile they want with their hostname. The function utilizes iPixie's shell scripting environment to create these options, gathering valuable system information and event logging in the process.

7.0.458 Technical description

- Name: ipxe_host_install_menu
- Description: The function creates an installation option menu for users to configure their host settings in regards to Thin Compute Host, Storage Cluster Host, or Disaster Recovery Hosts. It then logs the selected menu, and processes the menu item.

- Globals: None.
- Arguments: This function does not directly take in any arguments. However, variables like TITLE_PREFIX, FUNCNAME, CGI_URL, mac are processed internally within the function.
- Outputs: An install menu with log message and selection options.
- Returns: No return value as it's not needed in bash scripting, but it generates a host install menu for users to interact with which is the intended effect.
- Example usage:

  ```
  ipxe_host_install_menu
  ```

7.0.459 Quality and security recommendations

1. Input Validation: Validate input from users when configuring their hosts to ensure they are within permissible and expected values to avoid unforeseen issues.
2. Error Handling: Although the function attempts to log when a failure occurs (i.e., `echo Log failed`), significantly more robust error handling could be beneficial. This can include implementing logging for all failure points or at least planning for recovery or alternative steps for when failures occur.
3. Security Improvements: Avoid the exposure of sensitive data in log messages. For example, IP addresses, host identifiers, or any potentially exploitable system information.
4. Documentation and Comments: Include proper in-line comments to provide a better understanding of the script. Furthermore, each argument, even if supplied indirectly, should have a detailed explanation, making the script more maintainable and understandable to other developers.

7.0.460 `ipxe_host_install_sch`

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

Function signature: 76d8d63df923bfeaa3d9b18625e12e7909180d679baee564a7d5760de6c84baa

7.0.461 Function Overview

The `ipxe_host_install_sch` function is used to provide a menu for installing a Storage Cluster Host (SCH). It applies an iPXE menu pull script that will query the user for installation options, print the associated messages, and execute the chosen installation path. This function will immediately execute the installation after user selects the option.

7.0.462 Technical Description

- **name**: `ipxe_host_install_sch`

- **description**: This bash function creates an interactive iPXE menu for configuring a Storage Cluster Host (SCH) installation. The user is provided with options for ZFS single-disk or ZFS RAID installation.
- **globals**: `TITLE_PREFIX: represents the prefix of the menu title`, `CGI_URL: represents the URL to fetch and chain commands`
- **arguments**: This function does not require any arguments.
- **outputs**: The function prints an iPXE menu for user interaction.
- **returns**: The function does not have a return value but alters the control flow of the application based on user selection.
- **example usage**:

```
ipxe_host_install_sch
```

7.0.463 Quality and Security Recommendations

1. Check for the availability of global variables before using them. This will prevent unexpected outputs.
2. Verify your endpoint `${CGI_URL}` is secure and trusted to prevent possible injection attacks.
3. Avoid using plaintext for critical log messages or consider adding encryption for log messages to increase security.
4. It's recommended to handle possible failed fetch requests when `imgfetch` is unable to fetch data.
5. Consider adding validation for user input to avoid potential security risks.

7.0.464 `ipxe_network_boot`

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

Function signature: 12978609d623d739eeb7768560e3b7ba0c3cd5d711ae46c9949271f4abc20919

7.0.465 Function Overview

The function `ipxe_network_boot()` is primarily designed to determine the type of host system and perform a network boot accordingly. The function fetches the host type and logs it for debugging purposes, before proceeding to inspect the host type and perform operations correspondingly. If the host type is 'TCH', it validates the Alpine repository before attempting to boot. If the validation fails, it logs an error message, sends a failure message, and returns 1, demonstrating unsuccessful execution. If validation is successful, a boot operation specific to 'TCH' systems is performed. For any other type of host, the method logs a message notifying that network boot is not supported for that host type.

7.0.466 Technical Description

- **Function Name**: `ipxe_network_boot`
- **Description**: Responsible for determining the host type and carrying out a network boot operation accordingly.
- **Globals**: `mac`: the mac address of the host system.
- **Arguments**: None
- **Outputs**: Logs messages for debugging, errors, and host type support. May execute a failure response if the Alpine repository is not prepared.
- **Returns**: `1` if the Alpine repository validation fails, otherwise no explicit return.
- **Example Usage**:

  ```
  ipxe_network_boot
  ```

  Note: This function could only be invoked without any parameters.

7.0.467 Quality and Security Recommendations

1. Consider including an input validation to check if a `mac` global is defined before performing any other operations.
2. Additional error handling could be implemented to account for any potential pitfalls or unexpected issues during the booting process.
3. Providing additional support for other host types beyond 'TCH' could enhance flexibility and universality of the function.
4. Regularly review and update the function to ensure compatibility with updated versions of the Alpine repository and host configurations.
5. Uphold good practice of ensuring the sensitive information, such as MAC addresses, used within the logs and outputs is secured appropriately.

7.0.468 `ipxe_reboot`

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

Function signature: c2e2fb28eaa8f9898d8b5cb181b2b278c884005d1a98813c38797c3225f12b52

7.0.469 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.

7.0.470 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"`

7.0.471 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.

7.0.472 `ipxe_show_info`

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

Function signature: cbd14427c29a67a77d52cb0fd250b99f45cb94110f4d4b0a3f1123ec4fe219a4

7.0.473 Function overview

The `ipxe_show_info()` function is mainly used to display different configurations of a particular host. This function pulls up information such as the host's IP, hostname, plat-

form, UUID, serial number, product, cluster configuration, and HPS (High Performance Storage) paths.

The function achieves this by taking a single argument, which determines the category of information that will be displayed for the host. The categories include `show_ipxe`, `show_cluster`, `show_host`, and `show_paths`.

7.0.474 Technical description

- **Name**: `ipxe_show_info()`
- **Description**: This function displays specific information about a host, based on what category is passed as an argument.
- **Globals**:
  - `HPS_CLUSTER_CONFIG_DIR`: The directory where the cluster configuration is stored
  - `CGI_URL`: Link to CGI (Common Gateway Interface) script which is used to implement the command `process_menu_item`
  - `HPS_CONFIG`: The file where the HPS (High Performance Storage) configuration is stored
- **Arguments**:
  - `$1: category`: The type of information to display about the host. The options are `show_ipxe`, `show_cluster`, `show_host`, and `show_paths`.
- **Outputs**: Information about the host in the requested category, such as IP address, hostname, platform, UUID, serial and product numbers, and more.
- **Returns**: Does not return a value.
- **Example usage**: `ipxe_show_info show_ipxe`

7.0.475 Quality and security recommendations

1. Always ensure variable sanitization before using any variable in command substitution. This removes potential harmful commands being hidden as a value for a variable.
2. Always quote the variables. This will prevent word splitting and pathname expansion.
3. Implement error handling for when file reading or command chain replacement fails, or when an unknown item category is passed as an argument.
4. Implement checking for edge cases where the category argument is not passed at all.
5. The `ipxe_show_info` function is essentially storing, processing, and displaying sensitive information about a host. It is recommended to add necessary security measures to protect this sensitive data from potential breaches.
6. Use `printf` instead of `echo` for better string handling, especially while displaying file contents.

7. Consider using local variables. This can help in preventing variable clashing and accidental modification of environment variables which can have impact on how the system functions.

7.0.476 `_is_tty`

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

Function signature: 53fbf6cc003bc7d9b4614fc9d372f3471ed01447874f4db99da2816eb3cb7e69

7.0.477 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).

7.0.478 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
```

7.0.479 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.

7.0.480 `keysafe_cleanup_expired`

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

Function signature: 3206e50a2d55d74e70a68067e5cb3e041c93edf9f4d93411da00ecaa4c3535d8

7.0.481 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.

7.0.482 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
  ```

7.0.483 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.

7.0.484 `keysafe_handle_token_request`

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

Function signature: c84ad9d04dca1d7aa2bb79eee2302c774f89404291568bc324a376f3d3c8f37d

7.0.485 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.

7.0.486 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"

7.0.487 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.

7.0.488 `keysafe_issue_token`

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

Function signature: a535c705c6f66c5f6bb4cb03f77f0a1320e62fa27e687ab3cf0e601508b20886

7.0.489 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".

7.0.490 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"`

7.0.491 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.

7.0.492 `keysafe_validate_token`

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

Function signature: cabc1a12f681dbd43df7a58a8cdb46b267a8bb8a11fcda90cb24e16e5ad3a669

7.0.493 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.

7.0.494 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"
```

7.0.495 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.

7.0.496 `list_cluster_hosts`

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

Function signature: be089e20c3a7eb6f1c96e5243449b6bffb67b4c25d9062593c07c1c4f6b55ffb

7.0.497 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.

7.0.498 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`

7.0.499 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.

7.0.500 `list_clusters`

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

Function signature: 3f954969c054c0715b3cb4975cc16f60b59455fed57d76db287682a09738b747

7.0.501 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.

7.0.502 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.

7.0.503 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.

7.0.504 list_local_iso

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

Function signature: 2551c5d34fea622a4876a48c635391772020f951a812cc21a423cc6b87227a67

7.0.505 Function Overview

The list_local_iso function in bash is designed to locate and list ISO files that match a specific naming pattern in a designated directory. The naming pattern is created based on the input arguments detailing the CPU type, manufacturer, operating system name, and optionally, the operating system version. If no matching ISO files are found, the function outputs an alert message and returns a status code of 1. If matches are found, the base name of each matching ISO file is outputted.

7.0.506 Technical Description

- Name: list_local_iso

- Description: The function searches for and lists ISO files in a directory that match a specified naming pattern, made up from the arguments for CPU type, manufacturer, operating system name, and potentially, the operating system version.
- Globals: None required.
- Arguments:
  - `$1`: The CPU type descriptor.
  - `$2`: The manufacturer descriptor.
  - `$3`: The operating system name descriptor.
  - `$4`: (Optional) The operating system version descriptor.
- Outputs: A status message indicating the search process for local ISOs and the names of any ISO files found that match the naming pattern. If no matches are found, an alert message is outputted.
- Returns: If no matching files are found, the function returns `1`.
- Example Usage: `list_local_iso intel dell windows 10`

7.0.507 Quality and Security Recommendations

1. Conduct regular checks on the permissions assigned to the directory referenced in this function to ensure ISO files are not accessible to unauthorized parties.
2. Sanitize inputs to prevent potential code injections or file misdirections.
3. Implement error handling or exception management for situations where the directory does not exist or is not accessible.
4. Improve the clarity of existing function documentation especially with regards to its behavior when handling optional arguments.
5. Consider enhancement to allow for different patterns or multiple directories searching.

7.0.508 `_log`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: f7f5d16961fc9339682891b49f2fefad3611a12964b1a3cd095d11a46c108959

7.0.509 Function overview

This Bash function, `_log()`, works as a logging subroutine that outputs a given string to both the console and a remotely configured server. It belongs within a larger script, and it's specific use case relates to logging messages within a ZFS pool creation workflow.

7.0.510 Technical description

- **Name:** `_log`

- **Description:** This function performs logging operations to both a remote log and the system console (if LOG_ECHO environment variable is set to 1). It is specifically designed for logging operations regarding the creation of ZFS pools on free disks.
- **Globals:** [LOG_ECHO: An environmental variable used to control if the logging output should also be printed to console.]
- **Arguments:** [$*: A variable-length list of arguments to be logged. These arguments are usually messages related to the operations performed in the ZFS pool creation process.]
- **Outputs:** Printed statements are sent to the console (if LOG_ECHO is set to 1) and the `remote_log` function.
- **Returns:** Nothing directly, but it outputs strings to the console and the remote logging service.
- **Example Usage:** `LOG_ECHO=1 _log "ZFS pool created successfully"`

7.0.511 Quality and security recommendations

1. Always sanitize the inputs before logging to prevent log injection attacks.
2. Ensure that the `remote_log` function correctly handles connection failures and other errors to prevent disruption of the main program.
3. Consider adding timestamp and log level (info, warning, error, etc.) information to the log messages to provide better logging context.
4. Regularly rotate and archive logs to prevent them from occupying too much disk space.
5. Encrypt sensitive data in logs to protect them from being exposed to unauthorized users.
6. Consider implementing rate limiting to prevent DOS attacks via rapid, repeated calls to the `_log` function.

7.0.512 `make_timestamp`

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

Function signature: 7ea1fc9d3621ad0a04879323d75cd0a5f1aa2468bf98b9b3c83ff2b66dfa8e3d

7.0.513 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.

7.0.514 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
```

7.0.515 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.

7.0.516 `mount_distro_iso`

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

Function signature: 9ee707a09f131340e40ead1764695e3c9db4201a8c595ad06c9a5f7028376217

7.0.517 Function Overview

The `mount_distro_iso` function is used to mount an ISO file of a Linux distribution. The function takes a string as an argument that represents the name of the Linux distribution. It checks for the presence of the ISO file in a predefined directory. If the ISO file exists, the function then proceeds to check if the ISO file is already mounted. If not, the function creates a mount point directory and mounts the ISO file to it.

7.0.518 Technical Description

> **name**: `mount_distro_iso` **description**: This function is used to mount an ISO file of a Linux distribution to a predefined directory based on the passed distribution string. **globals**: `HPS_DISTROS_DIR`: This is a reference to the directory where the distribution ISO files are stored. **arguments**: `$1 - DISTRO_STRING`: The name of the Linux distribution, `$2 - iso_path`: The full path to the Linux distribution ISO file **outputs**: Logs information about the status of the ISO file and its mount point. **returns**: Returns 1 if the ISO file is not found, or if the mount point already exists, else it does not return any value. **example usage**: `mount_distro_iso ubuntu`

7.0.519 Quality and Security Recommendations

1. Incorporate comprehensive error handling and validation checks. This will help in ensuring that the mounted ISO file is legitimate and not corrupted.
2. Make sure to properly sanitize the `DISTRO_STRING` input. This will protect against potential code injection risks.
3. Document the expected values for each input variable. This would help prevent errors when the function is used by other team members.
4. Use a more descriptive name for the `hps_log` function. This would make the function more understandable to other developers who might read the code.
5. Write unit tests for this function to ensure it behaves as expected under different scenarios.

7.0.520 `n_build_apk_packages`

Contained in `lib/host-scripts.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: b3e3f65534cf6a07816a44e1e4e8df3379dc968dcd2cadc15d78c450ebbecb93

7.0.521 Function overview

The function `n_build_apk_packages` is used to build APK packages for *OpenSVC server* and *OpenSVC client* extensions. It first checks for required environment variables, verifies if the package directories exist and then continues with building the server and client packages respectively. Upon successful build, the function locates these packages and reports their metadata - providing their size and location. Lastly, the packages are copied into the HPS packages directory.

7.0.522 Technical description

- *name*: n_build_apk_packages

- *description*: The function checks for required environment variables, verifies if package directories exist, builds server and client APK packages, locates the packaged files, copy them into predefined directory, and finally write a log entry stating the successful completion.
- *globals*: [OPENSVC_SERVER_PKG_DIR: Directory for opensvc server-package, OPENSVC_CLIENT_PKG_DIR: Directory for opensvc client-package, OPENSVC_PACKAGE_BASE_DIR: Base directory for housing packages, OPENSVC_VERSION: OpenSVC version for which the packages are being built]
- *arguments*: [None]
- *outputs*: Error, confirmation, and status messages printed on the standard output.
- *returns*: 0 on success, 1 if any error occurred
- *example usage*: It doesn't take any argument, so just call the function as it is in your shell script like `n_build_apk_packages`

7.0.523 Quality and security recommendations

1. Error messages could be sent to stderr rather than stdout to improve the separation of regular and error output.
2. Permission checks for critical files and directories should be added - The script doesn't check if it has write permission to the target directories.
3. Consider enhancing error handling - The script carries on with the build process even if the checksum generation fails.
4. The function calls 'cd' repeatedly, which can fail. If it does, subsequent parts of the script can have unintended effects.
5. Use more robust methods for changing directories such as `pushd` and `popd`.
6. Always double-quote your shell variable expansions to prevent word-splitting and pathname-expansion.
7. Using `eval` can pose a potential security risk. Consider alternative ways to obtain the desired information.
8. Add more granularity to return codes so that different errors return different codes. This can make debugging easier.

7.0.524 `n_build_opensvc_binaries`

Contained in `lib/host-scripts.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: 7cc90027c9d9a353b0cd467113ad5b2a624efc0d5c66c5eb4a1ce7cbd73ec71b

7.0.525 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.

7.0.526 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`

7.0.527 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.

7.0.528 `n_build_opensvc_package`

Contained in `lib/host-scripts.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: 22c7385e472169b5941896aee8c618de46e294a277f781fa9db14d57d7bcef04

7.0.529 Function Overview

The `n_build_opensvc_package` function is an advanced installer script function designed to download, compile, and package the OpenSVC software into APK packages for Alpine Linux. OpenSVC is an open source project for managing and orchestrating heterogeneous application stacks within datacenters.

This function accepts several optional command-line parameters, including the Alpine version, specific OpenSVC git tag, and an option to keep the build directory after the process completes.

Without these, the function auto-detects the Alpine version and selects the latest compatible OpenSVC version based on the installed Go version.

7.0.530 Technical Description

Name: `n_build_opensvc_package`

Description: This function downloads, compiles, and packages the OpenSVC software into an Alpine APK package. Run it on a system with network connectivity, and adjust the target Alpine version, the OpenSVC version, or whether to keep the build directory via command-line arguments.

Globals:

- `alpine_version`: Target Alpine version (auto-detects if not specified)
- `om3_version`: Specific OpenSVC git tag (Chooses the latest compatible version if not specified)
- `keep_build`: Flag to keep the build directory after the process completes.
- `source_dir`: The source directory where the script will clone the OpenSVC repository.

Arguments:

- `$1`: Options: `--alpine_version`, `--om3_version`, `--keep_build`, `--help`
- `$2`: If `$1` is `--alpine_version` or `--om3_version`, `$2` should be the version or tag.

Outputs:

- Echoes build steps and relevant status information throughout execution
- In the case of a successful build, the script produces APK packages in the build directory.

Returns:

- `0` on successful execution and completion of all build tasks
- `1` on failure or error

Example Usage:

```
$ n_build_opensvc_package --alpine_version v3.7 --om3_version 2.0
 ↪  --keep_build
```

7.0.531 Quality and Security Recommendations

1. In the `--help` section, expand the option descriptions for better understanding of the function capabilities.
2. Use safer constructs for argument parsing, such as `getops`, which provides more robustness and flexibility.
3. Apply proper error handling. Instead of only exiting with `return 1` upon errors, it would be beneficial to give more varied return codes depending on the error type.
4. Provide a logging system that logs notable events and possible errors.
5. Where feasible, avoid running the script as root to minimize potential security risks.
6. Keep all code up to date with the latest safe practices and actively audit & review the code, patching any discovered insecurities.

7.0.532 `n_check_build_dependencies`

Contained in `lib/host-scripts.d/alpine.d/BUILD/01-install-build-files.sh`

Function signature: abe6f9e3a232a393f27bf444712eae8038b19089d61694e69a5f394e185e3f68

7.0.533 Function Overview

The `n_check_build_dependencies` function is used for checking available build dependencies of a software. It tests the presence of listed commands on the local system. If an unavailable command is detected, it alerts the user and suggests an install command along with adding the name to a list of missing packages that will be logged remotely. If all commands are present, the function returns a success message.

7.0.534 Technical Description

- **Name:** n_check_build_dependencies
- **Description:** This function checks the presence of required build dependencies by executing each as a command and checking the system's response. The absence of any required command is logged and if any are missing, a command is suggested to install the missing packages.
- **Globals:** None
- **Arguments:** None
- **Outputs:**
  - Prints the status of each required command (Present or Missing).
  - If any packages are missing, prints the names of the missing packages and provides a command to install them.

  - In case of missing commands, logs the missing packages remotely and alerts about the failure of build environment check.
  - If all commands are present, alerts about the successful build environment check.
- **Returns:** Returns 1 if any dependency is missing, 0 if all present.
- **Example usage:**

```
source n_check_build_dependencies.sh
n_check_build_dependencies
```

7.0.535 Quality and Security Recommendations

1. Implement input validation or sanitization to enhance security. For instance, avoid code injection by making sure commands and package names are not injectable.
2. Handle possible cases of failure more effectively. For example, if the internet connection fails during downloading or installation of packages.
3. Consider implementing a way to specify alternative packages for different distributions. This function assumes use of the apk package manager which is not available on all distributions.

7.0.536 `n_check_go_version_compatibility`

Contained in `lib/host-scripts.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: aeb6eb34265b6239b1da565dd0e2cd2ec192fa90cfc1f26a9ae963bcf59afdc2

7.0.537 Function overview

The given function, `n_check_go_version_compatibility`, checks if the currently installed Go version is compatible with the Go version requirement specified in a particular git tag of a source repository. The source repository is located at `/srv/hps-resources/packages/src/opensvc-om3` by default. If the installed Go version is lesser than the required version, the function will return an error stating the incompatibility. If no Go version is required or Go isn't installed, corresponding warnings and errors will be returned.

7.0.538 Technical description

- **Name**: `n_check_go_version_compatibility`
- **Description**: This function checks the installed Go version against the required Go version for a specific git tag in a source repository, and reports whether the installed version satisfies the required one.
- **Globals**: None
- **Arguments**:
  - `$1`: Name of the Git tag to check Go version requirement (desc)

  - $2: Not used in the function
- **Outputs**: Printed statements for errors, warnings or status of compatibility for Go version
- **Returns**:
  - 1 if there's an error or if installed Go version doesn't meet the required version.
  - 0 if installed Go version is sufficient or if required Go version cannot be determined.
- **Example Usage**:

```
$ n_check_go_version_compatibility v1.2.3

Go version compatibility check:
  Required: 1.14
  Installed: 1.16
  Status: Compatible
```

7.0.539 Quality and security recommendations

1. Ensure that the source code directory is a secure location and permissions are correctly set to avoid unauthorized access or modifications.
2. Use descriptive error messages to allow better debugging and troubleshooting.
3. Check if the git tag exists before proceeding with the rest of the function to avoid unnecessary operations.
4. Use secure methods to execute shell commands, to prevent command injection vulnerabilities.
5. Function relies on local environment's Go installation and source code directory. These dependencies should be documented and managed properly.
6. The function assumes that Go versions only consist of major and minor versions. This might not be the case always, need to handle sub-minor versions as well.

7.0.540 `n_clone_or_update_opensvc_source`

Contained in `lib/host-scripts.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: 52f6718d72bd0134e7d189fdc858d3e63ae616bcecdc33d4cbf81fb3466beb9e

7.0.541 Function overview

This shell function is designed to clone or update the OM3 module of the OpenSVC framework from a designated GitHub repository. It's named `n_clone_or_update_opensvc_source()`, and it fetches code from `https://github.com/opensvc/om3`. If a local copy exists, it pulls the latest updates; if not, it clones the repository.

7.0.542 Technical description

- **Name:** `n_clone_or_update_opensvc_source`
- **Description:** This function clones or updates the OpenSVC OM3 module from its GitHub repository.
- **Globals:** None
- **Arguments:** None
- **Outputs:** Logs and error messages.
- **Returns:** `1` if an error occurs (e.g., if the source directory exists but isn't a valid git repository, if a parent directory fails to create, if unable to clone the repository, or if unable to fetch updates), `0` if the function succeeds.
- **Example usage:** `n_clone_or_update_opensvc_source`

7.0.543 Quality and security recommendations

1. It's essential to make sure only trusted sources are used when cloning repositories for robust security.
2. During any git operations, ensure that the repository's source is secure.
3. Run regular security audits on the repositories to prevent the introduction of malicious software.
4. Implement error handling for all external operations, such as database queries.
5. When generating directories and handling files, confirm that their permissions are correctly set to avoid unintentional access from unauthorized users.

7.0.544 `n_configure_minimal_networking`

Contained in `lib/host-scripts.d/alpine.d/networking-functions.sh`

Function signature: 905f4f54d7e9ce8973823b4bbf269c4358c4830871e5290d9228cc2a31beed11

7.0.545 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.

7.0.546 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`

7.0.547 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.

7.0.548 `n_configure_motd`

Contained in `lib/host-scripts.d/alpine.d/console-control.sh`

Function signature: d87371da5d1ef4cf5f7025f69387b9e18a089f696a2ece496bc5166f820d9398

7.0.549 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.

7.0.550 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`

7.0.551 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.

7.0.552 `n_configure_reboot_logging`

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

Function signature: 28c7377daa009a1321d4d2192c2bf12eac35c875f5548a03265f2613661bb03b

7.0.553 Function Overview

The `n_configure_reboot_logging` function configures the logging for the reboot process on a TCH (Thomas Cook Holidays) node. It outputs status messages, creates directories, fixes broken symlinks for commands such as `reboot`, `poweroff`, and `halt`, creates wrapper scripts, handles shutdown, sets PATH, adds the OpenRC shutdown hook, and finally verifies the set up. The function makes use of `n_remote_log` and `n_remote_host_variable` functions to remit logs to the remote endpoint and set variables on the remote host. This function is primarily used to keep a track of the reboot process in a distributed system scenario and is an integral part of system maintenance and debugging procedures.

7.0.554 Technical Description

```
name: n_configure_reboot_logging
description: This function configures reboot logging on a TCH
 ↪  node. It makes sure that /usr/local/sbin exists then it fixes
 ↪  any broken symlinks for the commands reboot, poweroff, and
 ↪  halt. The function creates wrapper scripts for each command.
 ↪  The function handles shutdown, ensures /usr/local/sbin is
 ↪  first in PATH, adds the OpenRC shutdown hook and finally
 ↪  verifies the setup.
globals: []
arguments: []
outputs: Logs to the remote endpoint and sets variable on the
 ↪  remote host on the success or failure of each operation within
 ↪  the function.
returns: This function always returns 0 after successful
 ↪  operation. Any error condition needs to be reliably resolved
 ↪  within the function itself.
example usage: This function is called without any parameters.
 ↪  Call it like this: `n_configure_reboot_logging`.
```

7.0.555 Quality and Security Recommendations

1. The function can be made more robust by adding error checks after each operation and returns or logs in case of failure. Maximum reliance on successful operation may lead to silent failures.
2. Instead of hardcoding paths like /sbin, /usr/local/sbin in the script, they should be stored in variables at the beginning of the function. This would make the function more versatile and easier to maintain.
3. The function lacks file and directory permissions checks. For example, in some cases, the function might fail due to missing permissions to create directories,

remove or create files. Proper permissions checks should be added before file and directory manipulation operations.

4. Although grep ensures that PATH modification is not duplicated in `/etc/profile`, it is recommended to add a check for `/usr/local/sbin` is in PATH before adding to avoid any potential duplication.
5. It is recommended to test the existence of `n_remote_log` and `n_remote_host_variable` before using them. If these functions are not available, the function should return an error or fallback to local logging or variable setting.
6. The focus should be on reducing global and environmental side effects. For example, modifying PATH globally in `/etc/profile` might have unforeseen side effects. This should be well documented and considered in the implementation and usage of the function.

7.0.556 `n_console_message`

Contained in `lib/host-scripts.d/pre-load.sh`

Function signature: 36e26683aca03d37f7320e5524aa23849cd906d839f1064363a2204b560d0b17

7.0.557 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.

7.0.558 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.

7.0.559 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.

7.0.560 `n_create_apk_package_structure`

Contained in `lib/host-scripts.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: 14d2b37fdc223e4e9159f02ef271c3a1da681e0619b2284a916d0b907bcef8e0

7.0.561 Function overview

The `n_create_apk_package_structure` function in Bash is designed for creating a structure for an APK package. This function works by first checking the required environment variables to determine the OpenSVC version and build directory. The version is then transformed following the Alpine format. It then verifies if the build directory exists and if the binaries are available within it. If these condition are met, it then detects the Alpine version before creating the basic directories for the APK package structure.

7.0.562 Technical description

- **name**: n_create_apk_package_structure
- **description**: A Bash function designed to create a structure for an APK package using OpenSVC version, OpenSVC build directory, and the Alpine version.
- **globals**: [$OPENSVC_VERSION: OpenSVC version, $OPENSVC_BUILD_DIR: Directory to build OpenSVC]
- **arguments**: N/A
- **outputs**: Statements displaying the creation process and the package structure including version, APK version, Alpine, and package directory. It also displays error messages when required environment variables aren't set or when certain directories or files do not exist.
- **returns**: 1 if any errors occur (i.e. missing environment variable, erroneous directory/files)
- **example usage**: To use this function, simply call it in your script like so:

```
n_create_apk_package_structure
```

7.0.563 Quality and security recommendations

1. For improved security, consider using more precise error handling instead of a general return 1 statement for different types of errors.
2. Always ensure that your environment variables are secured and not easily accessible which could pose a security risk.
3. Check permissions on directories and files to ensure only authorized users can access the data.
4. Consider adding more comments to increase code readability and maintainability in the long term.
5. Include checks or validations to ensure the version strings are valid.

7.0.564 `n_disable_getty_alpine`

Contained in `lib/host-scripts.d/alpine.d/console-control.sh`

Function signature: 343d39eecf990fe5be10497769cdb8aa77d7ae96ed497cb2761fd51e47cac66c

7.0.565 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`.

7.0.566 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`.

7.0.567 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.

7.0.568 `n_display_info_before_prompt`

Contained in `lib/host-scripts.d/alpine.d/console-control.sh`

Function signature: 973863ced9e380918f2a6cfb0d3cb925aab9f7171b50b1c0b6920877dcdb4010

7.0.569 Function Overview

The `n_display_info_before_prompt()` function is used predominantly to display node information on the console before the prompt. This function first checks if the console is enabled or disabled. If the console is disabled, it simply logs the information and returns. If the console is enabled, it proceeds to ensure the `n_node_information` command is available before executing further commands. Then, it creates an issue file that displays before logging in and ensures it's readable. Finally, the function displays the issue file immediately on the console and logs the creation of the issue file with node info. The function returns 0 indicating successful execution.

7.0.570 Technical Description

- **Name:** `n_display_info_before_prompt()`
- **Description:** This function displays node information in the console before the prompt arises.
- **Globals:** None
- **Arguments:** None
- **Outputs:** Logs statements like "Displaying node information on console", "Console is disabled - display will start via inittab", and "Created /etc/issue with node info". It also displays content of issue file on the console.
- **Returns:** 0, indicating successful execution of the function.
- **Example Usage:**

```
$ source <path-to-script-containing-function>
$ n_display_info_before_prompt
```

7.0.571 Quality and Security Recommendations

1. Although the function is encapsulated to avoid global variables, it uses a lot of system commands which may not be safe from all forms of shell-injection attacks. It is recommended to use safer forms of commands or sanitize user input.
2. Robust error handling is not present. It is recommended to add more condition-based returns for errors and implement broader exceptions.
3. While the function checks whether certain commands exist before trying to run them, there is no fallback method if these commands are not available.
4. Function lacks explicit commenting on various sections, making it hard to understand for newer developers.
5. The function could be modularized further for code clarity and function reusability. For instance, separate functions for checking if the console is disabled and creating the issue file could be beneficial.

7.0.572 `n_enable_console_output`

Contained in `lib/host-scripts.d/pre-load.sh`

Function signature: e0d095431c3b9587d08666c0618947e69ed05b49e76a2626b8c52d77ed672e8f

7.0.573 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.

7.0.574 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
```

7.0.575 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.

7.0.576 `netmask_to_cidr`

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

Function signature: 591afba929cc6e4793e6bae6dbae1c9f2b4f4429ff9102867ddd8a651f7f496d

7.0.577 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`.

7.0.578 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
```

7.0.579 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.

7.0.580 `network_calculate_subnet`

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

Function signature: 28bad9be5b146b636db38a5bc9383098acfa045e1d0212500adddc909c91161d

7.0.581 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.

7.0.582 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
```

7.0.583 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.

7.0.584 `n_force_network_started`

Contained in `lib/host-scripts.d/alpine.d/networking-functions.sh`

Function signature: 448c7f84e1898821d351326c82f995cb0da05d48164042795fd8f9ec14ea1b86

7.0.585 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.

7.0.586 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.

7.0.587 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.

7.0.588 `n_force_start_services`

Contained in `lib/host-scripts.d/alpine.d/KVM/install-kvm.sh`

Function signature: 7f43681f2f170c1a4dd9b133c2e8539a06c8e5b262636f6cb4d83f6dc823b233

7.0.589 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.

7.0.590 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
```

7.0.591 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.

7.0.592 `n_get_provisioning_node`

Contained in `lib/host-scripts.d/pre-load.sh`

Function signature: 4e8d2d80a56db706bb5ebb500956281db15dd8b0c1afadab8b2a07d2b711d176

7.0.593 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.

7.0.594 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"
```

7.0.595 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.

7.0.596 `n_install_apk_packages_from_ips`

Contained in `lib/host-scripts.d/alpine.d/install-packages.sh`

Function signature: 38f25d6a4707f41d280cc4eda71c9dc104b5f7d91cc99be723b18ccbd8081c71

7.0.597 Function overview

This Bash function `n_install_apk_packages_from_ips()` is used to install apk packages from an IPS server. The function accepts a list of package names as arguments, fetches the corresponding packages from the specified IPS server, and installs them. In the process, it logs various events, such as the start and end of the installation process, any problems encountered during the process, and a log of successfully installed packages.

7.0.598 Technical description

- **Name:** n_install_apk_packages_from_ips

- **Description:** This function is responsible for fetching and installing apk packages from an IPS. An error message is thrown when the package list is null. Upon successful execution, it logs the successful installation or any encountered errors.
- **Globals:** None
- **Arguments:** ($@) an array of package names to be installed.
- **Outputs:** Various messages and logs that indicate the progress of the operation.
- **Returns:** 0 if all packages are successfully installed; 1 if no packages are provided or encountered errors during the process; 2 if it fails to create a temporary directory or to install the packages.
- **Example usage:** `n_install_apk_packages_from_ips package1 package2 package3`

7.0.599 Quality and security recommendations

1. The function could be improved by validating the package names before attempting to install. For example, check the package names against a list of available packages and displaying a user-friendly error message for invalid names.
2. Avoid the use of temporary directories in the root (`/tmp`) directory. This is a commonly used directory and naming conflicts could potentially occur.
3. Handle edge cases where the IPS gateway cannot be determined.
4. Currently, an unsuccessful package fetch doesn't cancel the operation. It would be better if the function stopped on the first error encountered, instead of continuing to attempt to fetch other packages.
5. The curl commands should have better error handling, to deal with cases where the server may be unavailable, slow, or returned an error response.
6. Implement a rollback feature, where in case of any error during the installation of packages, the changes made should be rolled back to keep the system state consistent.
7. The function should have better exception handling to capture and handle any errors that are encountered during its execution.
8. Consider using HTTPS for the repo URL to enhance security.

7.0.600 `n_install_base_services`

Contained in `lib/host-scripts.d/alpine.d/BUILD/05-install-utils.sh`

Function signature: 4cd6b1fb7ba56a5274df7644a0c4fa2b3d40481f53d79993030a5e76b148313b

7.0.601 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.

7.0.602 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`

7.0.603 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.

7.0.604 `n_install_kvm`

Contained in `lib/host-scripts.d/alpine.d/KVM/install-kvm.sh`

Function signature: 2c9f3c8070fa470d9f3db678b80540e2d69be579379d0a0539a928a932844ed9

7.0.605 Function Overview

The `n_install_kvm` is a function that is used to install and setup the KVM virtualization environment on a host. It handles the installation of necessary packages, enabling necessary services at boot, and checking for the proper functioning of the network. It also confirms whether the system is in boot sequence and holds off starting the libvirtd service until the completion of the boot sequence. If the installation or setup fails at any

point, it logs this error and sets the appropriate `virtualization_status` for further troubleshooting.

7.0.606 Technical Description

- **Name:** `n_install_kvm`
- **Description:** This function installs and sets up the KVM virtualization environment on a host. It logs all steps and error conditions and manages the process based on the system's boot sequence and network status.
- **Globals:** None
- **Arguments:** None
- **Outputs:** Function outputs a series of status messages and logs the installation steps as well as any errors to a remote log. It also sets remote host variables to reflect the installation status and the type of virtualization installed.
- **Returns:** Returns 0 if installation and setup succeed. Returns 1 if the package installation fails and 2 if the libvirtd service fails to start.
- **Example usage:** `n_install_kvm`

7.0.607 Quality and Security Recommendations

1. In order to prevent some potential security vulnerabilities, ensure all input to the function is properly sanitized.
2. Ensure that the function is being run with the correct permissions. Elevate privileges only when necessary to avoid possible privilege escalation attacks.
3. Instead of silencing error messages, log them to a secure, centralized location for effective troubleshooting and security incident response.
4. Regularly update the packages being installed by the function to get the latest security patches.
5. Implement rigorous error handling and validate all command outputs to avoid unexpected issues and system disruptions during the installation and setup process.

7.0.608 `n_interface_add_ip`

Contained in `lib/host-scripts.d/common.d/n_network_functions.sh`

Function signature: 7e40d3cc8158f30bbc3b61ed12b5bdee7c2150fd7411dc6b70f9cb96ed29457e

7.0.609 Function overview

The `n_interface_add_ip` function in Bash is used to add an IP address with an optional netmask to a specified network interface. If the netmask is provided not as a CIDR, the function also converts it to CIDR format.

7.0.610 Technical description

- **Name:** `n_interface_add_ip`
- **Description:** This function is used to add an IP address to a network interface. If a netmask is provided not as a CIDR, the function converts it to the CIDR format.
- **Globals:** []
- **Arguments:**
  - `$1: iface` Network interface to which an IP address will be assigned.
  - `$2: ip_addr` The IP address to be added to the network interface.
  - `$3: netmask` The netmask for the IP address to be added. It can be in CIDR or standard netmask format.
- **Outputs:** The function outputs the return status of the `ip addr add` command.
- **Returns:** The function returns the exit status of the `ip addr add` command. If the command is successful it returns 0, otherwise, it returns a non-zero status.
- **Example usage:**

```
n_interface_add_ip eth0 192.168.1.10 24
```

7.0.611 Quality and Security Recommendations

1. It is recommended to add input validation for arguments to ensure correct values for network interface, IP address and netmask are passed to the function.
2. To ensure good security practices, you could check that the network interface exists before trying to add an IP address to it.
3. To ensure the good quality of function, consider the addition to handle error cases and outputs corresponding error messages to standard error for easier troubleshooting.

7.0.612 `n_ips_command`

Contained in `lib/host-scripts.d/pre-load.sh`

Function signature: 7c976d9bd3444a3ae8e13ed21d05f2c2e37ddef020e9be6b7dae46388db13c60

7.0.613 Function Overview

The `n_ips_command` function is a utility to send a POST request to an access point. The request will target the `boot_manager.sh` script hosted on the access point HTTP server. The requested command and parameters are embedded in the URL.

7.0.614 Technical Description

- **Name:** `n_ips_command`
- **Description:** Send a POST request to the IP address fetched from a function `n_get_provisioning_node` directing towards `boot_manager.sh` script as

a part of the URL. The function sends command as URL parameters and catches HTTP errors and Curl errors, providing error handling. Output from the curl command is parsed and HTTP error codes are checked. In case of no errors, the HTTP response is echoed.

- **Globals:**
  - `N_IPS_COMMAND_LAST_ERROR`: Variable used to save the error message in case of failure.
  - `N_IPS_COMMAND_LAST_RESPONSE`: Variable used to preserve the last HTTP response.
- **Arguments:**
  - `$1`: command string to be sent as a POST request, REQUIRED.
  - Subsequent arguments: key-value pairs as additional parameters, OPTIONAL.
- **Outputs:** Either `N_IPS_COMMAND_LAST_ERROR` and `N_IPS_COMMAND_LAST_RESPONSE` assigned with error codes and HTTP response upon failure, or HTTP response itself upon success.
- **Returns:** Error codes upon HTTP or curl errors or 0 upon successful HTTP request.
- **Example usage:** `n_ips_command "print" key=value anotherkey=anothervalue`

7.0.615 Quality and Security Recommendations

1. Include more descriptive error messages for easier debugging.
2. Secure the POST request by adding some form of authentication to prevent unauthorized access.
3. SSL/TLS could be utilized to add encryption to the communication.
4. Better handle potential edge-cases, avoid possible command injection by escaping special characters.
5. Make error codes consistent or adjustable through variables to maintain compatibility and flexibility if used in larger workflow.

7.0.616 `n_keysafe_request_token`

Contained in `lib/host-scripts.d/common.d/keysafe-node.sh`

Function signature: 43c0b2dc4a1a9560d4393853037cc9eaf34c1e671a67475620b7086e407cea75

7.0.617 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.

7.0.618 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"`

7.0.619 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.

7.0.620 `n_load_kernel_module`

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

Function signature: e8d3e185dcbaf9a04014e264ef088ba7b3adf4b74b818ea01d0f4f6ddc3c93d9

7.0.621 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.

7.0.622 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`

7.0.623 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.

7.0.624 `n_load_remote_host_config`

Contained in `lib/host-scripts.d/pre-load.sh`

Function signature: deea1cfd1cac05c37febb11a35389234fe97871c7dce4258577047d2511a289a

7.0.625 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.

7.0.626 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`

7.0.627 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.

7.0.628 `n_network_find_best_interface`

Contained in `lib/node-functions.d/common.d/n_network-functions.sh`

Function signature: 30cff9d21045991e6d995dbe891f122f2351ae05e020c8176afdf81e3ca77061

7.0.629 Function Overview

The `n_network_find_best_interface` function in Bash is designed to find the fastest network interface that matches the specified minimum speed and state requirements. It first sets `min_speed` and `req_state` variables, then iterates through all network interfaces and checks these variables against each interface's current state and speed. The function keeps track of the fastest interface that matches the requirements, and finally outputs the name of this interface, if any, without a newline.

7.0.630 Technical Description

- **name**: `n_network_find_best_interface`
- **description**: This Bash function takes two optional arguments, `min_speed` and `req_state`, which default to 0 and "up" respectively if not provided. It finds the fastest network interface that matches these requirements and prints the name of this interface to standard output.
- **globals**: None.
- **arguments**:
  - `$1: min_speed` - The minimum speed requirement.
  - `$2: req_state` - The required state of the network interface ("up", "down", or "any").
- **outputs**: The name of the best network interface that meets the specified requirements is printed to standard output.
- **returns**: Nothing.
- **example usage**:

```
n_network_find_best_interface 100 "up"
```

7.0.631 Quality and Security Recommendations

1. Robustly handle edge cases such as network interfaces with names that contain special characters or white spaces.
2. Better error handling, for instance when no network interface is found to meet the requirements.
3. Additional security considerations may be in place to prevent potential command injection during the network interfacing process.
4. Unit tests for function stability.
5. Consider implementing logging for when expected conditions are not met.
6. Try to avoid global variables where possible to prevent unintended side effects.

7.0.632 `n_network_select_storage_interface`

Contained in `lib/node-functions.d/common.d/n_network-storage-functions.sh`

Function signature: d1f3c7ca9e0f0e6929baa09c1e6832503572a02c3ef5c01c419be9172a3ed7bc

7.0.633 Function Overview

The function `n_network_select_storage_interface` is designed to identify and select the most optimal network interface for storage. It first attempts to find a 10 Gigabit or faster interface. If such an interface is not found, it will then select the fastest interface that is currently available. The selection made by the function is logged remotely and then echoed for the user to see. This function returns `0` if an interface is selected and `1` if no interface is found.

7.0.634 Technical Description

- **Name**: `n_network_select_storage_interface`
- **Description**: This function finds the best network interface for storage, first trying to find a 10G+ interface. If not found, the fastest available interface is selected. The function logs the result remotely and echoes back to the user.
- **Globals**: None
- **Arguments**: None
- **Outputs**: The name of the selected network interface or nothing if none found.
- **Returns**: `0` on success (i.e., an interface is selected), otherwise it returns `1`.
- **Example usage**: `n_network_select_storage_interface`

7.0.635 Quality and Security Recommendations

1. Sanitize any input that is provided to prevent potential security vulnerabilities.
2. Handle all possible error scenarios to ensure the function does not crash or produce unpredictable outcomes.
3. Test the function in different network conditions to validate its performance and reliability.
4. Avoid hard-coding values like '10000' and '0'. Instead, use configuration settings or parameterize the function.
5. Implement logging to record function activities, which is crucial for diagnosing problems in the future.

7.0.636 `n_node_information`

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

Function signature: 74929d3d0eddb047d5e64379e90dffc1956cc56b3f72b1d2b31d9b0358c73d56

7.0.637 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.

7.0.638 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`

7.0.639 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.

7.0.640 `node_lio_create`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: 98eefecb076b12ce8574e70fbcdc3dbe7da279292f2b27fad96ad89b969a1f6e

7.0.641 Function overview

The `node_lio_create` function sets up an iSCSI target node using the Linux IO (LIO) Target-core storage abstraction layer. It does so by taking three optional arguments for iqn, device, and acl. The function validates and checks the existence of required arguments (iqn and device), extracts the backstore name from iqn, creates the backstore, the iSCSI target and the LUN, configures the ACL if provided and saves the configuration.

7.0.642 Technical description

- **name:** node_lio_create
- **description:** A bash function to set up an iSCSI target node using the Linux IO (LIO) Target-core storage abstraction layer via targetcli.
- **globals:** None
- **arguments:**
  - `$1`: iqn – the iSCSI Qualified Name, an unique identifier for an iSCSI node.
  - `$2`: device – the device to be used for the iSCSI target.
  - `$3`: acl – Access Control List to control which initiators are granted access.
- **outputs:** Logs messages regarding process status (e.g., success/failed to create backstore/iSCSI target/etc.)
- **returns:**
  - `0`: if the iSCSI target has been successfully created.
  - `1`: if a failure occurred (unknown parameter, missing required parameters, non-existing device, failure to create backstore/iSCSI target, etc.)
- **example usage:** `node_lio_create --iqn iqn.2003-01.com --device /dev/sdb --acl iqn.1994-05.com.redhat:dhclient`

7.0.643 Quality and security recommendations

1. The function should sanitise all user input data to prevent potential command injection attacks or faulty operations.
2. The function could implement some logging mechanism, preserving a history of operations for further analysis or debugging.
3. Improve error messages by stating exactly which parameter(s) is/are missing instead of the generic message
4. The function could handle different input formats and conversions. For instance, device input could be accepted as device UUID instead of device path only.
5. Including more comprehensive tests for edge cases or exceptional circumstances, such as when the system lacks the necessary resources to create a new target.
6. Encryption should be considered for the transmitted data as this function disables authentication for demo/testing purposes. This feature should be taken out of production versions or controlled by an additional variable.

7.0.644 node_lio_delete

Contained in lib/host-scripts.d/common.d/iscsi-management.sh

Function signature: 198653d7f8219c254d0f7d905c865d8994dd5dc9072af9cdbc979296660c9662

7.0.645 Function Overview

The function node_lio_delete() is used to delete an iSCSI target and associated backstore in a Linux system. It accepts an iSCSI Qualified Name (IQN) as an argument. This function safeguards by validating the provided argument, checks if the iSCSI target exists before attempting deletion, and similarly for the associated backstore. Post-deletion task involves saving the configuration.

7.0.646 Technical Description

```
Function: node_lio_delete
Description: Deletes an iSCSI target and its associated backstore.
Globals: None
Arguments:
    - $1: IQN (iSCSI Qualified Name) of the target to be deleted.
Outputs: Logs information about the execution process, which
 ↪  includes successful or unsuccessful deletion of the iSCSI
 ↪  target and the backstore.
Returns:
    - 0 on successful deletion of the target and backstore, or if
     ↪  they do not exist.
    - 1 if the parameter is unknown or missing, or if deletion of
     ↪  the target or backstore is unsuccessful.
Example Usage:
    node_lio_delete --iqn iqn.2003-01.org.linux-
     ↪  iscsi.localhost.x8664:sn.4afce5632cfd
```

7.0.647 Quality and Security Recommendations

1. Add a mechanism to validate the structure of the IQN argument. Validating the IQN increases security by preventing injection or incorrect usage.
2. Add error handling for the targetcli execution when executing commands like targetcli /iscsi delete "${iqn}". Execution success should not be implicitly assumed.
3. Incorporate additional logging for more insight into function execution process.
4. Include a mechanism to backup current configuration before making changes. Recovery from failure scenarios can be quicker.

7.0.648 node_lio_list

Contained in lib/host-scripts.d/common.d/iscsi-management.sh

Function signature: f36324bce9bac497425dd3ef1f0cb414f4b7fe0e640904c41e749f78c8e29a70

7.0.649 Function overview

The `node_lio_list` is a Bash function that lists out iSCSI targets and block backstores from a system. The function leverages the `targetcli` command for pulling and organizing the required information, thus providing an easy and direct interface for iSCSI and backstore management.

7.0.650 Technical description

Name: `node_lio_list`

Description: The function retrieves and displays the iSCSI targets and block backstores present in the system by using the `targetcli` command. It showcases the results one after the other for ease of visibility and understanding.

Globals: None

Arguments: None

Outputs: The function prints two lists. The first list displays the iSCSI Targets. This segment of the program executes the `targetcli` command to get the iSCSI targets. After a line break, the second list displays the block backstores also using the `targetcli` command.

Returns: It returns `0` after executing the `echo` commands to signify that the function executed successfully.

Example Usage:

```
node_lio_list
```

7.0.651 Quality and security recommendations

1. The function is relatively simple and does not handle errors. Users should be aware that any errors occurring in the `targetcli` command will break the function. Proper error handling should be considered to ensure resilience of the function.
2. The use of global variables could be considered for more customized output.
3. While not a problem in this specific function, keep in mind that usage of `echo` to output data can lead to potential command injection if user-provided data is not properly sanitized.
4. Always review your Bash scripts for potential security vulnerabilities. This could include examining how it handles input, reviewing the script for potential command injection vulnerabilities, and checking the script for insecure file operations.
5. Use helper functions to make the code reusable and modular. The function should do one thing and do it well. This function is a good example of that.

7.0.652 node_lio_manage

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: 86138e13e14957c8703d9fb6016f95dbd6eec12c63716fc419c9afcb1c48e6ca

7.0.653 Function Overview

The `node_lio_manage()` function is a controller function in a node's local I/O (LIO) management system. Using a case statement, it dispatches the given 'action' to the appropriate functions. Actions include 'create', 'delete', 'start', 'stop', 'status', and 'list'. If the input 'action' is not recognized, the function logs an error message and then returns with a value of 1.

7.0.654 Technical Description

- **name:** `node_lio_manage()`
- **description:** This function serves as a dispatcher for different actions of a node's local I/O (LIO) management system. The designated 'action' argument is checked against a predefined list ('create', 'delete', 'start', 'stop', 'status', 'list'), and upon match, the corresponding function is invoked.
- **globals:** None.
- **arguments:**
  - `$1`: Action to be performed on a node's LIO. Valid: 'create', 'delete', 'start', 'stop', 'status', 'list'.
  - `$@`: Options that may be provided following the 'action', to be passed on to the corresponding function.
- **outputs:** Logs a usage message if no 'action' is provided, or if the 'action' given is not recognized. Also dispatches tasks to various functions which may have their own outputs.
- **returns:** `1` for invalid 'action' or missing argument; the exit status of the last executed command otherwise (`$?`).
- **example usage:**

```
- node_lio_manage create
- node_lio_manage delete
```

7.0.655 Quality and Security Recommendations

1. Incorporate more stringent input validation (e.g., checking the format and/or value of the 'action', or other argument inputs, prior to usage).
2. Use more descriptive error messages and create separate logs for errors and usage to make the system easier to debug.
3. Implement specific security measures to protect against command substitution or code injection attacks.

4. Make sure that the user running the script has the necessary permissions for all functions that this one is dispatching to, without granting more permissions than necessary.
5. Test the function in various scenarios to ensure it behaves as expected, including cases of erroneous input.

7.0.656 `node_lio_start`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: 618a6b47c51365d7704455f712c3beca1d289fd4105a628552ebf67c3fbc5573

7.0.657 Function Overview

The Bash function `node_lio_start` is designed to start a "target service" in a Linux environment, using systemd's `systemctl` command. It does this through calling `systemctl start target`, which starts the target service, and `systemctl enable target`, which ensures that the service will be started at boot time. Log messages are sent to a remote server via a function called `remote_log` before starting the service and after either successfully starting the service or failing to do so. A success or failure status is indicated by returning 0 or 1 respectively.

7.0.658 Technical Description

Below is a technical description of the `node_lio_start` function:

- **name**: `node_lio_start`
- **description**: A Bash function designed to start a "target service" using systemd's `systemctl` command. Also ensures said service will be started at boot, and records the result (either success or failure) by sending a log message to a remote server.
- **globals**: None.
- **arguments**: None.
- **outputs**: Logs messages to a remote server indicating the process of starting and enabling the service.
- **returns**:
  - 0 if the service is successfully started and enabled.
  - 1 if either starting or enabling the service failed.
- **example usage**:

  ```
  node_lio_start
  ```

7.0.659 Quality and Security Recommendations

1. Always ensure the `remote_log` function is properly defined in the same shell environment where `node_lio_start` is called to avoid errors.
2. Implement error catching for case when `systemctl` command is unknown, i.e., systemd isn't in use.
3. Safeguard this function with user and permission checks to ensure only authorized personnel or scripts can initiate systemd services.
4. Incorporate status checks before starting the service to prevent launching an already running service.
5. Add more logging for better troubleshooting, such as logging the status code of the `systemctl` command.

7.0.660 `node_lio_status`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: 327a33e59e304acd75f9e12b0cd6fa4aca564708c04f3615da39246c5573ba39

7.0.661 Function overview

The function `node_lio_status()` is designed for the purpose of querying the status of a Target Service and displaying its LIO configuration. It uses Linux's `systemctl` and `targetcli` commands to retrieve and demonstrate this data.

7.0.662 Technical description

- **Name:** `node_lio_status`
- **Description:** The function echoes a status report for a target service and its LIO configuration utilizing the systemctl and targetcli commands respectively. The function doesn't take any parameters or global variables.
- **Globals:** None
- **Arguments:** None
- **Outputs:** The target service status report and LIO configuration data are printed to stdout.
- **Returns:** Always returns 0 signifying that the function has successfully completed its task.
- **Example usage:** `bash node_lio_status`

7.0.663 Quality and security recommendations

1. It would be beneficial to make sure that the required commands (`systemctl` and `targetcli`) are available in the system before actually utilizing them in the function.

2. The command outputs might require interpretations which can be hieroglyphic or error-prone to a non-technical user. It might be advantageous to include added verbosity/reasoning to the echoed content for the sake of usability.
3. Remember to check the return values of the commands and handle any errors that might occur. Currently, the function always returns 0 irrespective of whether the commands run successfully or not.
4. It's good to utf-8 sanitize the output for proper display and to prevent possible injection attacks. This is especially crucial if the output is intended to be utilized or displayed elsewhere.
5. Always validate that the function is being run with the appropriate permissions since commands like `systemctl` and `targetcli` often require superuser rights.

7.0.664 `node_lio_stop`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: 971a38bb22f277cd0531cfa279368d8f7e4995e9a385bb6221485f8a90cc3bcd

7.0.665 Function Overview

The function `node_lio_stop` is a bash script function designed to stop a specific service, named target, running on a system. It logs its activity using another function `remote_log` and uses the `systemctl` command to stop the service. If successfully stopped, it generates a success message and returns 0. If the attempt to stop the service fails, it generates a failure message and returns 1.

7.0.666 Technical Description

- **Name:** node_lio_stop
- **Description:** This function attempts to stop the 'target' service on a system. It employs the '`systemctl` command to accomplish this, but also provides logging functionality through the `remote_log` function for tracking activity.
- **Globals:** None.
- **Arguments:** None.
- **Outputs:** This function generates log messages indicating the status of the stop operation - either successful or failed.
- **Returns:** This function returns a binary response. It returns 0 if the stop operation is successful and 1 if it fails.
- **Example Usage:**

```
node_lio_stop
```

7.0.667 Quality and Security Recommendations

1. Consider implementing error logging or notifications, such as sending an alert email, when the service fails to stop.
2. For enhanced security, you could use the `sudo` command to enforce the function to run with root permissions only, thus restricting the usage of the function to authorized users only.
3. Include verbose commenting, particularly for areas handling failures or errors, which can help for easier troubleshooting in future.
4. Consider allowing the function to accept a parameter to specify the service to stop, instead of hardcoding 'target'. This would make it more flexible and reusable.
5. Implement a check to understand whether the 'target' service is active before attempting to stop it.

7.0.668 `node_storage_manager`

Contained in `lib/host-scripts.d/common.d/common.sh`

Function signature: 739f69b5a6036d980bc51405a34e0ec8a34bed5236ca2984db5c0d656a80c5c6

7.0.669 Function overview

The `node_storage_manager` function manages storage components on a node. By using two main arguments, a component and an action, it decides which storage management function to call. The user can also optionally pass further arguments for specific storage management tasks. Component options include "lio" and "zvol," corresponding to different subsystems in the node's storage system. Actions can include operations like 'start', 'stop', 'status', etc. The function validates the arguments, dispatches to the appropriate function based on the given component, logs the execution and handles any errors.

7.0.670 Technical description

node_storage_manager

- **Description**: This function manages storage components for a machine node. It logs an error and returns 1 if any of the main arguments is missing. The function then checks the value of the 'component' argument and calls the corresponding function for 'lio' or 'zvol'. If the 'component' argument doesn't match any of these, it logs an error and returns 1. The function then captures the exit status of the called function, logs a respective message and returns the result.
- **Globals**: None
- **Arguments**:
  - `$1`: The component to manage. Can either be 'lio' or 'zvol'.
  - `$2`: The action to execute on the given component.

  - `...`: Optional arguments passed to the dispatched function.
- **Outputs**: Logs messages to the remote log regarding the action being performed, its success or failure, and its return code.
- **Returns**: 0 if the dispatched function was successful, 1 if the arguments are invalid or the dispatched function failed.
- **Example usage**: `node_storage_manager lio start`

7.0.671 Quality and security recommendations

1. Refrain from passing sensitive data as arguments or using them in log messages.
2. Add more detailed error logging to help with troubleshooting potential errors.
3. Add input sanitization to the function to prevent possible shell injection attacks.
4. Consider adding more explicit validation logic for argument values to catch common user input errors.
5. Be sure that called functions 'node_lio_manage' and 'node_zvol_manage' are implemented securely and robustly.
6. Protect all logging data, especially if it contains sensitive information.
7. Use specific return codes for different errors to help users better understand why the function failed.
8. Document any modifications made to the function for clarity.

7.0.672 `node_zvol_check`

Contained in `lib/host-scripts.d/common.d/zvol-management.sh`

Function signature: 07447af71573f33e1e9896011244fa50252bca03c45d1685a0b8677a4720d46c

7.0.673 Function Overview

The `node_zvol_check` function is a part of the command-line tool intended for use in ZFS (Zettabyte file system). The function checks if a specific ZFS volume (zvol) exists, where the zvol is defined by its pool and name. These two parameters are passed to the function as arguments. The function prints the outcome of the check to the console and returns an error code.

7.0.674 Technical Description

- **Name:** `node_zvol_check`
- **Description:** This function checks the existence of a ZFS volume (zvol) where the zvol is identified by its ZFS pool and name.
- **Globals:** No globals variables
- **Arguments:**
  - `$1`: Parameter pair `--pool <pool>`.
  - `$2`: Parameter pair `--name <name>`.

- **Outputs:**
  - If the zvol exists, it prints: `Zvol <zvol_path> exists`.
  - If the zvol does not exist, it prints: `Zvol <zvol_path> does not exist`.
- **Returns:** `0` if the zvol exists; `1` otherwise, or if the required arguments were not correctly passed.
- **Example usage:** `node_zvol_check --pool myPool --name myZvol`

7.0.675 Quality and Security Recommendations

1. Implement parameter validation, ensuring the input parameters meet the expected format.
2. Add more explicit error codes for different error conditions to aid in troubleshooting.
3. Implement a help message function when parameters are not provided or incorrectly supplied.
4. Consider using more secure methods when processing the output and display messages. This could help against possible Bash Injection attacks.

7.0.676 `node_zvol_create`

Contained in `lib/host-scripts.d/common.d/zvol-management.sh`

Function signature: ec6e430fb8789091cfe8fdd4b1e197a7a20d2619e9cb9986827d740d74268de1

7.0.677 Function Overview

The Bash function `node_zvol_create()` is designed to create a ZFS (Zettabyte File System) volume, or zvol. This is done through the use of environment parameters, namely a pool, name and size, which are parsed and passed as variables to the system's `zfs` command. The function validates the provided parameters and checks for the existence of the zvol before attempting to create a new one.

7.0.678 Technical Description

- **Name**: `node_zvol_create`
- **Description**: This function creates a ZFS volume using given parameters. The parameters are parsed for three variables: pool, name, and size. The function then uses these variables in the `zfs create` command to generate the volume, first checking to ensure that it does not already exist.
- **Globals**: None
- **Arguments**:

    - $1, $2: These are used by the command to parse the given parameters. Each shift command removes the current value of $1 and assigns the next parameter value to it.
- **Outputs**: conditional logging to a remote destination detailing the function's activity and success or failure to create a volume.
- **Returns**:
    - 0: Success - zvol was created without issue.
    - 1: Failure - Either due to an unrecognized parameter, missing required parameters, or pre-existing zvol.
- **Example Usage**:

```
node_zvol_create --pool tank --name vol1 --size 1G
```

7.0.679 Quality and Security Recommendations

1. Enforce stricter validation for the 'pool', 'name', and 'size' parameters to prevent any potential command injection attacks or inconsistencies.
2. Implement robust error handling to make the function more resilient to unexpected behavior and to give more informative responses when something goes wrong.
3. Use unique and clear logging statements to make the progress of the function easier to follow and troubleshoot.
4. Store sensitive data securely to ensure this data isn't exposed to unauthorized access.
5. For improved auditability, incorporate logging for all major events such as input validation failures, zvol creation failure, and zvol creation success.
6. Find ways to limit the use of global variables to lessen the chances of unwanted side effects from their modification.
7. Whenever possible, encapsulate complex sections of code into their own functions to improve readability and maintainability.

7.0.680 `node_zvol_delete`

Contained in `lib/host-scripts.d/common.d/zvol-management.sh`

Function signature: 21fe8b803dc5e01613ebe91d459c8ff30011593e6a87412bc6122f867aeec227

7.0.681 Function overview

The `node_zvol_delete` function is designed to handle the deletion of ZFS volumes (`zvol`). The function accepts two parameters: the name of the ZFS storage pool and the name of the specific volume. It first checks whether these parameters have been provided or not. If they are missing, it will return an error message and exit. If they are provided, it will construct the path to the ZFS volume and then check whether that volume exists. If it doesn't, it will return an error message and exit. If it does, it will

attempt to delete the volume and return a success message if successful or an error message if not.

7.0.682 Technical description

The following properties describe the function in a technical manner:

- **Name:** `node_zvol_delete`
- **Description:** Deletes a node's ZFS volume
- **Globals:** None
- **Inputs:**
  - `--pool`: The pool parameter (a type of ZFS storage)
  - `--name`: The name of the volume to be deleted
- **Outputs:** Messages indicating the process of deletion and its success or failure
- **Returns:**
  - `0` if the volume is successfully deleted
  - `1` if there's an error (either because of missing parameters or failure in deletion)
- **Example usage:**

```
node_zvol_delete --pool mypool --name myvolume
```

7.0.683 Quality and security recommendations

1. Add more in-depth input validation for both `--pool` and `--name` parameters to ensure they conform to expected formats or value ranges.
2. Implement error handling that doesn't merely log and return but also performs some sort of troubleshooting or recovery operation.
3. Consider adding an optional force-delete parameter to allow bypassing certain checks when necessary.
4. Ensure that all output messages, especially error messages, do not leak sensitive system state information.
5. Implement strict access control measures to protect against unauthorized zfs pool and volume deletion.

7.0.684 `node_zvol_info`

Contained in `lib/host-scripts.d/common.d/zvol-management.sh`

Function signature: 80d5ff3413c64a25ffdb5394756a3fcb79290986c8d4a3bfd18cdf837a95a543

7.0.685 Function Overview

This function `node_zvol_info()` is a bash function primarily used to fetch and display ZFS volume (zvol) information from a specified pool. It accepts themed arguments,

`--pool` and `--name`, to determine the pool and the name of the volume, respectively. Once the parameters are provided, it validates their existence, then proceeds to check if the zvol exists in the system. Should all these checks pass, it displays the zvol information—its name, volume size, usage, available space, and reference association —along with its device path.

7.0.686 Technical Description

- **Name:** node_zvol_info
- **Description:** This function checks specific zvol in the provided zfs pool and name, returns its existence status and displays the zvol information if exists.
- **Globals:** Remote_log:logs for the remote server.
- **Arguments:**
  - `$1: --pool`: The name of the ZFS storage pool that contains the ZFS volumes.
  - `$2: --name`: The name of the volume to be checked in the pool.
- **Outputs:** The function prints zvol information to stdout, which includes name, volume size, usage, available space, and references; additionally, it provides the exact device path.
- **Returns:** It returns 1 if there's a failure (invalid/unspecified parameters, or non-existant pool/volume). It returns 0 if it successfully found the zvol and displayed its status.
- **Example Usage:**

```
node_zvol_info --pool tank --name volume1
```

7.0.687 Quality and Security Recommendations

1. Implement more robust error handling. This could include more specific error messages depending on the type of error encountered.
2. Use secure methods when handling the arguments to prevent potential command injection attacks. Escaping or quoting variables can help in this regard.
3. Enhance the logging process. Log all the errors to a specific error log file with timestamps. This will be helpful for debugging purposes if anything goes wrong.
4. Refrain from displaying too much detailed information to unprivileged users. This can be used to gather information for potential attacks.
5. Implement a usage function or comprehensive help menus to assist users in case of syntax confusion or misuse.

7.0.688 `node_zvol_list`

Contained in `lib/host-scripts.d/common.d/zvol-management.sh`

Function signature: 95f86f7af006e06fa8df84870e8f7eff728b79dd9bcb448fd864b4d7b350276b

7.0.689 Function overview

The `node_zvol_list` function is primarily used to list zvols (ZFS volumes) that exist either in a specified ZFS pool or in the entire system.

7.0.690 Technical description

- **Name:** `node_zvol_list`
- **Description:** This function lists all the zvols present either in a specified pool or in the entire system depending upon the argument provided. It logs any unknown parameters and errors associated with the listing of the zvols.
- **Globals:** `pool`: Holds the name of the pool wherein zvols are to be listed.
- **Arguments:** `[ --pool "$2" ]`: Optional argument where $2 is the name of the pool.
- **Outputs:** This function will output the list of zvols along with their volume size on stdout.
- **Returns:** If the function successfully lists the zvols, 0 (zero) is returned, otherwise 1.
- **Example usage:**

```
node_zvol_list --pool my-zfs-pool
```

7.0.691 Quality and security recommendations

1. Sanitize input: Always check and sanitize the inputs provided by the user to avoid any sort of injection or overflow attacks.
2. Error handling: The current function only returns 1 when an error condition is met. However, different error codes can be returned for different error situations to improve error understanding and debugging.
3. Usage of variables: Avoid using global variables as it might lead to unexpected results if multiple functions are modifying them at the same time.
4. Documentation: Maintain proper comments and explanation for the different parts of the function. This will help other developers understand the functionality quickly and accurately.
5. Functional scope: Be clear about the function scope. A function named “node_zvol_list” should ideally only list node zvols and not do anything else. This will maintain the function’s purity and make it easier to debug and maintain.
6. Exit early: If a conditional check fails, exiting the function as soon as possible can enhance the performance and readability of the code.

7.0.692 `node_zvol_manage`

Contained in `lib/host-scripts.d/common.d/zvol-management.sh`

Function signature: fbc75475f9ed8b4e90c9fe423b227bbd08f041872fcaf46c3bee9ac215f09937

7.0.693 Function overview

`node_zvol_manage()` is a function that acts as a controller for the operations related to managing ZFS volumes (Zvol). Necessary operations such as `create`, `delete`, `list`, `check` and `info` are handled based on the `action` argument passed by the user. In case no `action` is provided or an invalid `action` is passed, the function logs an error message and returns.

7.0.694 Technical description

- **Name:** `node_zvol_manage`
- **Description:** This function manages ZFS volumes (Zvol). It handles different operations like creation, deletion, listing, checking, and access information of Zvols. These operations are carried out based on the `action` argument provided to the function. If an invalid `action` is given, it logs an error message and returns.
- **Globals:** None
- **Arguments:**
  - `$1: action` - specifies the operation to be performed on the Zvols.
- **Outputs:** Executes one of the Zvol operations (`create`, `delete`, `list`, `check`, `info`), or logs an error message in case of invalid action.
- **Returns:**
  - `1` if no action or an invalid action is provided.
  - Exit status of the Zvol operation otherwise.
- **Example usage:** `node_zvol_manage create [options]`

7.0.695 Quality and security recommendations

1. Provide more specific error messages to help identify issues quickly.
2. Safeguard function against invalid number of arguments beyond the `action` argument depending upon the operation.
3. Consider handling operations asynchronously to improve performance and efficiency.
4. Incorporate additional security measures especially while deleting or modifying Zvols to avoid accidental data loss.
5. Document usage of the function in detail to avoid any misuse or misunderstanding.

7.0.696 `normalise_mac`

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

Function signature: 597ad94d9559d604fae13c8fd8ef8bde5b6c19fee81c409691213ad492d3d6b3

7.0.697 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.

7.0.698 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`

7.0.699 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.

7.0.700 `n_prepare_build_directory`

Contained in `lib/host-scripts.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: da5d976a232d610c6dee7320c9a7dd5f46b21af2840c2356476910456cae90e6

7.0.701 Function overview

This Bash function `n_prepare_build_directory` is used to prepare a directory for building the Opensvc version specified by `OPENSVC_VERSION`. It verifies the existence of the source directory, creates a temporary build directory, makes sure the directory does not already exist, copies the source tree to the build directory, and sets up the Go build environment for static compilation.

7.0.702 Technical description

- **Name**: `n_prepare_build_directory`
- **Description**: Prepare directory for building OpenSVC. The function assumes that `OPENSVC_VERSION` and `source_dir` are set.
- **Globals**: [`OPENSVC_VERSION`: The version of OpenSVC to use for building, `OPENSVC_BUILD_DIR`: The directory to use for building OpenSVC, `CGO_ENABLED` : Flag indicating whether to use Go build environment for static compilation]
- **Arguments**: None
- **Outputs**: Warnings or errors if the source directory does not exist, the build directory exist already or the build directory is unable to be created. It also provides information on the build and the version.
- **Returns**:
  - 1 if `OPENSVC_VERSION` is not set, source directory does not exist or if it fails to create or delete the build directory
  - 0 if the build directory has been prepared successfully
- **Example usage**:

```
export OPENSVC_VERSION=2.0
n_prepare_build_directory
```

7.0.703 Quality and security recommendations

1. Consider making the `source_dir` a script parameter or a global configuration so as to avoid hard coding paths within functions.
2. It might be better to notify the user that the build directory is being deleted. They may have files in there which they didn't realize would be this function's responsibility.
3. To prevent potential directory traversal or file overwrite attacks, ensure that the value of `OPENSVC_BUILD_DIR` is controlled or validated.
4. The error messages printed by this function could be standardized and written in a dedicated error handling function, to make them easier to manage and keep the codebase DRY (Don't Repeat Yourself).
5. Run static analysis tools to catch more nuanced potential security issues such as format string vulnerabilities, buffer overflows etc.

7.0.704 n_queue_add

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

Function signature: fb031729b3dc0f645733e0bf0019f6e8a5f9a8814aaf402d9ae00f483110e1d8

7.0.705 Function overview

The function `n_queue_add()` is designed to add commands into a queue, which is managed in the form of a file defined by the global variable `N_QUEUE_FILE`. This function accepts an arbitrary number of arguments, which are supposed to represent a single shell command with its arguments. When run, it logs error messages to `stderr` in case no function was specified or if it failed to append the command to the queue file. Successful queuing of the command is also logged but not to `stderr`.

7.0.706 Technical description

- **Name:** n_queue_add
- **Description:** Accepts a function call and appends it to the queue file denoted by global variable “N_QUEUE_FILE”. It logs error messages if the function call is empty or in case of failure and success messages otherwise.
- **Globals:** [N_QUEUE_FILE: Represents the file used as a queue for commands]
- **Arguments:** [$*: Represents the function call to be added to the queue]
- **Outputs:** Logs an error message to stderr if no command was specified or if it fails to add the provided command to the file. If the command was successfully added to the file, it log the command along with the message stating its successful addition.
- **Returns:**
  - 1: When the function call is empty or when an error occurs while adding to the queue file.
  - 0: When the command is successfully added to the queue file.
- **Example usage:**

```
n_queue_add echo "Hello, World!"
```

7.0.707 Quality and Security Recommendations

1. Validate commands before adding to the queue, to verify that only expected and safe commands are ran.
2. Implement checks for sufficient disk space before writing to the queue file to avoid a potential space exhaustion.
3. Consider adding rate limiting or max queue capacity to prevent flooding of the queue.
4. Sanitize inputs to avoid potential code injection risks.

5. Log all operations, not just errors, to help in identifying unusual activities or patterns.

7.0.708 n_queue_clear

Contained in lib/node-functions.d/common.d/common.sh

Function signature: 46874d549f99a809edba232d4d4449fa033e053884a353edbb632b7b0c84c641

7.0.709 Function Overview

This function, n_queue_clear(), is used for clearing the function queue. It does this by deleting the queue file stored in N_QUEUE_FILE and logging the action using n_remote_log function with the message "Function queue cleared". It returns 0 upon successful execution.

7.0.710 Technical Description

- **Name:** n_queue_clear
- **Description:** This function clears the function queue by removing the queue file and logging the action.
- **Globals:**
  - N_QUEUE_FILE: Represents the queue file which needs to be deleted.
- **Arguments:** This function does not accept any arguments.
- **Outputs:** It logs the action of clearing the function queue.
- **Returns:** 0 upon successful completion.
- **Example Usage:** bash n_queue_clear

7.0.711 Quality and Security Recommendations

1. For enhanced security, ensure that the file in N_QUEUE_FILE has appropriate permissions set, so that unintended users can't modify it.
2. Usages of rm -f can be dangerous as it forcefully deletes files without confirmation. Ensure the variable N_QUEUE_FILE is correctly set and the deletion is intended.
3. Always handle the error scenarios. Right now, even if the removal of the queue file fails, the function would return 0 which signifies successful execution. Proper error checking and corresponding return value should be set.
4. It is recommended to have meaningful logging messages and including these logs to a dedicated log file would be useful for future debugging.

7.0.712 n_queue_list

Contained in lib/node-functions.d/common.d/common.sh

Function signature: e8abb5da44c7c785c4bd046616ed2ff8bdc38e921596af163d9b256f8c921b1b

7.0.713 Function Overview

The function `n_queue_list()` is a Bash script function that is designed to list queued functions in an order of their addition to the queue. It reads from a file specified by the environment variable `N_QUEUE_FILE`, calculates the total number of queued functions and then prints each function number and its name. In case the file does not exist or there are no functions queued, a respective message is issued.

7.0.714 Technical Description

- **Name**: `n_queue_list`
- **Description**: This function lists all the queued function calls. It uses the total variable to count the number of function calls and the variable i to index them. If the `N_QUEUE_FILE` doesn't exist or is empty, it will echo "No functions queued". Otherwise, it will echo each function call along with its index in the queue.
- **Globals**: [`N_QUEUE_FILE`: `N_QUEUE_FILE` is an environment variable that stores the name of the file that contains the queue.]
- **Arguments**: [None. This function does not take any arguments.]
- **Outputs**: The function displays the total number of functions in the queue. For each queued function, it outputs the function number and its call. If no functions are queued or the `N_QUEUE_FILE` does not exist, a respective message is printed.
- **Returns**: The function always returns 0 which means it executed successfully.
- **Example Usage**:

```
N_QUEUE_FILE=function_queue.txt
n_queue_list
```

7.0.715 Quality and Security Recommendations

1. Sanitize inputs and ignore or properly handle unexpected characters in the contents of `N_QUEUE_FILE`.
2. Always provide clear and understandable messages about errors, especially when file does not exist or cannot be read.
3. Document that this function manipulates the environment variable `N_QUEUE_FILE`, which might have side effects if not properly controlled.
4. Be aware that race conditions or interrupted signals could impact this function, especially when reading and writing to files. Implement concurrency controls if necessary.
5. Add more error checking logic throughout the function.

6. Consider providing usage information or 'help' context within the script for better user understanding.
7. Limit the execution permissions only to the needed users/groups for better security.

7.0.716 n_queue_run

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

Function signature: 7aebf0010c5ac557d85e2e81d19f75eeaa26e75c3ea54996d34864da26ec7377

7.0.717 Function overview

The `n_queue_run` function is used to execute a queue of functions read from a specified queue file, logging remotely the starting execution and the results of each function call. It keeps track of total function calls, how many of them were successful and how many failed. After all function calls are processed, the queue is cleared and the function returns the number of failed executions.

7.0.718 Technical description

- **name**: `n_queue_run`
- **description**: This function servers to execute a queue of functions read from a specified queue file and log the outcomes of each function call, whether it was successful or failed.
- **globals**: `N_QUEUE_FILE`: the path to the file containing the queue of functions to be executed.
- **arguments**: No arguments are employed in this function.
- **outputs**: The function logs the beginning of the queue execution, the outcome of each function call in the queue and the completion of the queue. In the end, it also logs how many functions were successfully executed and how many failed.
- **returns**: The function returns the number of failed function executions, which could be 0 if all function calls were successful.
- **example usage**:

```
# Execute queued functions
n_queue_run
```

7.0.719 Quality and security recommendations

1. Ensure that the `N_QUEUE_FILE` is secured so that unintended users cannot modify the function queue.
2. Enforce a timeout for each function call so that potential infinite loops in the functions can be handled.

3. Consider introducing a mechanism to hold the last known “safe” state so that we can rollback if failure rate in the execution queue is high.
4. Make sure that your `n_remote_log` implementation is thread-safe if this function is intended to be used in multi-threaded programs.
5. In case of failure, consider adding a mechanism to retry execution a certain number of times before logging the failure and moving on.
6. Be cautious when using `eval`. Ensure that it is only used to execute trusted code.

7.0.720 `n_remote_cluster_variable`

Contained in `lib/host-scripts.d/pre-load.sh`

Function signature: 508394addef239d8b24fec9fbf55cb2cd13174f75881778577137eec6fbeecb8

7.0.721 Function overview

The function `n_remote_cluster_variable` is used to interface with a cluster, specifically to either set or get a variable. The function first checks if there are at least two arguments: the variable name and its value. If the function receives two arguments, it sets the variable with the provided name and value. If only the variable name is provided, it gets the value of the variable with that name. If the operation is successful, the function outputs the result and returns 0. Otherwise, it logs the error and returns the exit code.

7.0.722 Technical description

- **Name**: `n_remote_cluster_variable`
- **Description**: Sets or retrieves a variable in a cluster depending on the given input arguments.
- **Globals**:
  - **VAR**: Not applicable; there are no global variables used in this function.
- **Arguments**:
  - **$1**: The name of the variable for which the value is to be retrieved or set.
  - **$2**: An optional argument which, when present, is the value to be set for the variable.
- **Outputs**: Echoes the result of the operation if successful, or logs an error message if not.
- **Returns**: 0 if the operation is successful, returning the exit code otherwise.
- **Example usage**:

```
n_remote_cluster_variable "testVariable" "testValue"
```

7.0.723 Quality and Security Recommendations

1. There's no validation of input for variables $1 and $2, which could lead to unexpected behaviour or result in a security vulnerability. It's recommended to sanitize and validate the inputs for these variables.
2. Error messages should be more detailed, including the operation that failed, and if possible, the reason for the failure.
3. It's always good practice to comment on each block of the function. Proper commenting makes its understanding and debugging easier.
4. Global constants for error messages could be used to help homogenize the error handling across different functions.

7.0.724 `n_remote_host_variable`

Contained in `lib/host-scripts.d/pre-load.sh`

Function signature: a6a6dcce5193b038b3b72301649c925f02faeada882cdad35f26a54763120ec0

7.0.725 Function overview

The `n_remote_host_variable` function is used to get or set remote host variables. The function is built to receive two arguments: a `name` which refers to the name of the variables, and an optional `value`, which if present, signifies that we are performing a SET operation. If the `value` is absent, the function defaults to a GET operation. The function executes an `n_ips_command` to perform either the variable SET or GET operation, and receives an exit code. It outputs the result of the operation if it is successful, otherwise it logs the error description and returns the error code.

7.0.726 Technical description

7.0.726.1 name

`n_remote_host_variable`

7.0.726.2 description

Performs GET or SET operation on a specified remote host variable depending on the presence of a second argument.

7.0.726.3 globals

`N_IPS_COMMAND_LAST_ERROR`: stores the last error message from the `n_ips_command`.

7.0.726.4 arguments

`$1`: Specifies the name of the remote host variable.

`$2`(optional): If this parameter is present, a variable SET operation is performed. If not, a GET operation is executed.

7.0.726.5 outputs

If the operation is successful, the result from the `n_ips_command` operation will be printed. Otherwise, an error message will be logged.

7.0.726.6 returns

0: If the `n_ips_command` operation is successful.

Otherwise returns the code signifying the type of error occurred during the operation.

7.0.726.7 example usage

To get a remote host variable:

```
n_remote_host_variable "hostname"
```

To set a remote host variable:

```
n_remote_host_variable "hostname" "127.0.0.1"
```

7.0.727 Quality and security recommendations

1. Always ensure that the name parameter (`$1`) is provided while making a call to `n_remote_host_variable`. If not, the function should throw an informative error.
2. Ensure to validate user inputs before use. This can help ward off potential security vulnerabilities such as command injection attacks.
3. Make sure to always check the exit code of the `n_ips_command` to handle potential errors effectively.
4. It's recommended to enclose variables in double quotes to avoid word splitting and globbing.
5. Log all potential error messages for easier debugging.
6. Always remember to check the validity of the host variable before making a GET or SET operation call to the `n_ips_command`.

7.0.728 `n_remote_log`

Contained in `lib/host-scripts.d/pre-load.sh`

Function signature: 0f3f34eb844b248235362d1ec5c747e0d2e05f807738bdd13abc7927c1be4755

7.0.729 Function overview

The provided function `n_remote_log()` is used to log messages remotely. The input parameters to the function are a message and the function from which `n_remote_log()` is called. The function ensures the correctness of the inputs and validates the successfully processing of the `n_ips_command`. In the scenario where `n_ips_command` fails, the function outputs error information to the stdout and returns the corresponding exit code. If the `n_ips_command` executes successfully, the function simply returns 0.

7.0.730 Technical description

- **Name:** `n_remote_log`
- **Description:** This function logs messages to a remote server, taking in the message and the calling function as its arguments. It outputs errors to stdout and returns the exit code of the `n_ips_command` if it fails. If there is no failure, it returns 0.
- **Globals:**
  - `N_IPS_COMMAND_LAST_ERROR`: This stores the last error message if the `n_ips_command` execution fails.
  - `N_IPS_COMMAND_LAST_RESPONSE`: This stores the last response from the server if the `n_ips_command` execution fails.
- **Arguments:**
  - `$1: message`: The message to be logged remotely.
  - `$2: function`: The function that is calling `n_remote_log`.
- **Outputs:** Outputs error info to stdout if the `n_ips_command` fails.
- **Returns:** Returns the exit code of the `n_ips_command` in case of a failure, otherwise returns 0.
- **Example Usage:** `bash n_remote_log "Test message" "Test function"`

7.0.731 Quality and security recommendations

1. The function should handle the condition when the `message` is empty or when the `function` is not provided.
2. For better error handling, there should be different exit codes for different error scenarios. The function currently only returns the exit code of `n_ips_command` or 0.
3. In case of an error response from the server, additional information such as timestamp, server name, and error code would be beneficial for debugging.
4. Error messages should be logged in a separate error log, instead of stdout, to keep the stdout clean and improve error management.
5. Proper input sanitization and validation must be implemented to prevent command injection or code execution attacks.

6. Confidential information or sensitive data, if part of logs, must be handled appropriately or hashed before logging.

7.0.732 `n_select_opensvc_version`

Contained in `lib/host-scripts.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: 8d8072d2f9a864f354808032ebcb2f550aff33ca0eefa6db92379b9e2241998e

7.0.733 Function overview

This function, `n_select_opensvc_version()`, controls the version of the OpenSVC software by checking out a specified Git tag from a local copy of the OpenSVC source code. If no tag is provided, it defaults to the latest semantic version of the software.

7.0.734 Technical description

- **Name:** `n_select_opensvc_version()`
- **Description:** Checks out a specified Git tag for the source code, defaults to the latest semantic version if no tag is supplied. The function ensures the selected version is valid and the source repository is a valid Git repository.
- **Globals:** `OPENSVC_GIT_TAG:` The Git tag of OpenSVC being used, `OPENSVC_VERSION:` The version of OpenSVC being used stripped of the leading 'v'
- **Arguments:** `$1:` The version tag of OpenSVC that the user wants to select
- **Outputs:** Error messages if the source directory is not existent or if the requested git tag does not exist in the source directory. Also, exports the selected git tag and OpenSVC version globally, and outputs success message if the specified version is successfully chosen.
- **Returns:** 1 if invalid directory or invalid tag, else 0 after successful execution
- **Example usage:**

```
n_select_opensvc_version v2.0.7
```

7.0.735 Quality and security recommendations

1. In current form, this script relies on the assumption that any string supplied as $1 will be a valid Git tag. If this is not guaranteed, more comprehensive error checking should be added around this input.
2. Explicitly declare function scope of all variables to avoid accidental clash with global variables.
3. Ensure use of quoted variables to prevent word-splitting or globbing issues.
4. Regularly pull latest updates from the remote repository to make sure local copy is synchronized with the latest version.

5. Perform periodic repository health check and clean up unused tags to maintain optimal performance.

7.0.736 `n_set_hostname_and_hosts`

Contained in `lib/host-scripts.d/common.d/configure-hostname.sh`

Function signature: 5c49cc42685037b094c5b80248bbc1340247e3fb87f6b196b2249a61f999550f

7.0.737 Function Overview

The function `n_set_hostname_and_hosts` is a shell script utility to automate the process of setting the hostname and configuring the hosts file in a linux environment. It fetches the configuration values like hostname, domain, and IP from the function `n_remote_host_variable` and `n_remote_cluster_variable`. It sets the hostname using available methods (via hostnamectl or hostname command). It also creates or updates the file `/etc/hosts` with the appropriate hostname and IP entries.

7.0.738 Technical Description

- **Name:** `n_set_hostname_and_hosts`
- **Description:** This function sets the hostname, and updates the `/etc/hosts` file in a linux environment.
- **Globals:** None
- **Arguments:** No direct arguments, but uses outputs from `n_remote_host_variable` and `n_remote_cluster_variable` functions.
- **Outputs:** Messages to stdout, writes to /etc/hostname, /etc/hosts and logs via `n_remote_log`.
- **Returns:** Returns 1 if configuration fetch or validation fails, 2 if hostname change fails, 3 if /etc/hosts creation fails, and 0 on successful execution and completion.
- **Example usage:**

  ```
  n_set_hostname_and_hosts
  ```

7.0.739 Quality and Security Recommendations

1. Security: Restrict the permissions of this script to trusted users only in order to prevent unauthorized modifications of host configuration.
2. Robustness: Error handling seems to be in place for most of the function calls, further improvements can be made by having a catch-all error trap.
3. Validation: Additional validations can be added for the hostname, domain and IP values (like length, characters, format, etc.).

4. Advances: The function can be updated to handle more network configurations and to work on more types of Linux distributions, especially those that may not follow these conventions directly.
5. Logging: Error logs could include more specific details about which step or validation caused the error for easier troubleshooting.
6. Code-Reuse: The parts of the code for hostname settings and hosts file configuration, could be further broken down into more modular, re-usable functions.

7.0.740 `n_setup_build_user`

Contained in `lib/host-scripts.d/alpine.d/BUILD/01-install-build-files.sh`

Function signature: 58d6f11cd02c168aafff4c94b7ac6ad26ba376a01adf30d89d7457f348a020c8

7.0.741 Function overview

The `n_setup_build_user` function is used primarily for setting up a build user for APK package creation in a Linux environment. The function performs a series of tasks including checking if the specified build user exists, creating one if not, setting up a home directory for the user, ensuring the user is part of the 'abuild' group, creating a abuild signing key, and changing the permissions of a package directory.

7.0.742 Technical description

Name: `n_setup_build_user`

Description: A Bash function used for setting up build user for APK package creation. It performs several key operations such as creating a user, setting up home directory and group memberships and generating a signing key for the user.

Globals: 1. `build_user`: The name of the user that will be created for building the APK. Default value is 'builder'. 2. `packages_dir`: The directory on the system where APK packages are stored. Default value is '/srv/hps-resources/packages'.

Arguments: None

Outputs: The function outputs a running commentary of its actions as it executes, e.g. "Creating build user builder…", "Creating home directory…", etc.

Returns: - 1 if it fails to create the build user or to generate the signing key. - 0 after successfully setting up everything.

Example Usage:

```
n_setup_build_user
```

7.0.743 Quality and security recommendations

1. Consider parameterizing the function to accept the `build_user` and `packages_dir` as arguments, rather than hardcoding them. This will improve flexibility.
2. Proper error handling can be enhanced by including more explicit error messages in case a step fails and possibly exiting the function.
3. Security can be improved by checking the privileges of the script executing this function. It requires substantial system permissions (e.g., creating a user, changing the directory's ownership) that could lead to unintended security vulnerabilities.
4. Check the script for the possible command injections, as it is executing system commands. Ensure that all variables used in commands are sanitized and safe to use.

7.0.744 `n_show_vlan_interfaces`

Contained in `lib/host-scripts.d/common.d/n_network_functions.sh`

Function signature: 7be6dc67621a499ff331c0f30e602bda79e77117b88d6451d7e8f309ae8e9dbc

7.0.745 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.

7.0.746 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`.
```

7.0.747 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.

7.0.748 `n_storage_auto_configure`

Contained in `lib/node-functions.d/common.d/n_network-storage-functions.sh`

Function signature: 48aec0db7f247178adead2e76226ce0a07943a46af6470c04840cd451a937c09

7.0.749 Function overview

The function `n_storage_auto_configure` is used to auto-configure a storage with a given index and a preferable interface. If no preference is provided for the interface, an existing one is used. If the interface is down, it will be brought up. After getting an allocation from IPS, the VLAN is configured, and an IP is added accordingly. A quick connectivity test with the gateway is also implemented. If there are any issues during these steps, appropriate error messages get logged. The function returns 0 upon a successful run.

7.0.750 Technical description

- **Name:** `n_storage_auto_configure`
- **Description:** This function is used to auto-configure a storage network with a given index. If no preferable interface is provided, an existing one is selected. Connectivity issues and allocations are properly handled, and the necessary VLAN and IP settings are configured accordingly.
- **Globals:**
  - N/A
- **Arguments:**

  - $1: Storage index. Default value is 0.
  - $2: Preferable interface. If it's not provided, an existing selector would be used.
- **Outputs:** Logs helpful debugging messages such as "Configuring storage network" or "Bringing up interface", and more informative ones like "Storage network configured".
- **Returns:** Returns 1 when it encounters an error (no suitable interface for storage, failed allocation, failed VLAN creation, failed IP addition), 0 on success.
- **Example usage:** `n_storage_auto_configure 1 eth0`

7.0.751 Quality and security recommendations

1. Ensure error handling for all commands, not just for some critical ones like `n_vlan_create`. This is important in order to prevent sequential execution when the preceding command fails.
2. Validate the inputs before utilizing them in the function to avoid undesired commands or path manipulations.
3. Include proper logging. Currently, there's a good amount of logging taking place. Including more detailed logs might provide more insight in case of a failure.
4. Try to abstain or handle the use of command substition `$(...)` properly. It might pose a risk if not appropriately sanitized.
5. If not directly related to the function's purpose, separate parts of this function into distinct functions. This helps you increase reusability, reduce complexity, and improve maintainability.

7.0.752 `n_storage_network_setup`

Contained in `lib/node-functions.d/common.d/n_network-storage-functions.sh`

Function signature: 3faec5338d5de66aeb9bbdd4b525a1d39d507e8408bae022354f7a89bbc30b26

7.0.753 Function Overview

The `n_storage_network_setup` function in Bash is used to set up storage networking on a machine. It takes two optional parameters - a physical network interface and a storage index. If the physical interface is not provided, the function auto-detects it. The function uses the parameters to request IP allocation from the Intrusion Prevention System (IPS). If the allocation fails, the function returns an error and exits. Otherwise, it parses the allocation and configures the VLAN interface and IP address. Finally, it checks for network connectivity and logs the result.

7.0.754 Technical Description

Function Name: `n_storage_network_setup`

Description: This function sets up storage networking by creating a VLAN interface and configuring an IP address on it. It also tests network connectivity.

- **Globals**: [`phys_iface`: Physical network interface, `storage_index`: Storage index]
- **Arguments**: [`$1 (phys_iface)`: Physical network interface, `$2 (storage_index)`: Storage index (default `0`)]
- **Outputs:** Messages to standard output showing successful allocation of VLAN and IP, or errors in the process.
- **Returns:**
  - `0`: The function executed successfully
  - `1`: The function failed at any point due to an error
- **Example Usage:**

```
n_storage_network_setup eth0 1
```

7.0.755 Quality and Security Recommendations

1. Always validate the parameters coming to the function to prevent attacks and unexpected behaviors.
2. Handle the case where the IP allocation request returns an error other than "ERROR" at the start of the string.
3. Add better error messages and create error codes for different types of errors.
4. Always clean up resources that aren't necessary at the end of the function execution.
5. Use encrypted communication when dealing with network interfaces for security measures.
6. Ensure that the function handles failure of any intermediate commands gracefully.

7.0.756 `n_storage_validate_jumbo_frames`

Contained in `lib/host-scripts.d/common.d/n_storage-functions.sh`

Function signature: 88bd0071a499bbd8569aa662f5b5a8dc21749e5558c21d25b8da39efca395556

7.0.757 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.

7.0.758 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`

7.0.759 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.

7.0.760 `n_url_encode`

Contained in `lib/host-scripts.d/pre-load.sh`

Function signature: 4e21e5663f425634af5a4baa7396afcc1884c8ecdeb58e9f5ffd146304fd834a

7.0.761 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'.

7.0.762 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
```

7.0.763 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.

7.0.764 n_vlan_create

Contained in lib/host-scripts.d/common.d/n_network_functions.sh

Function signature: 3690b98c049b24b83d75f48dae821a680194f45f901cafdaffbaff34a2d58326

7.0.765 Function overview

The n_vlan_create function is used to create a Virtual Local Area Network (VLAN). A VLAN is a subnetwork within a larger network, providing network isolation at the data layer. In this function, an existing network interface is used to create a VLAN with a specified ID. The function first removes the VLAN if it already exists, then creates the VLAN and sets up the network interface with the desired Maximum Transmission Unit (MTU).

7.0.766 Technical description

- **name**: n_vlan_create
- **description**: Creates a VLAN with a specified ID for a physical network interface. If the VLAN already exists, it is removed before creating the new one.
- **globals**: None
- **arguments**:
  - $1: phys_iface (description: The name of the physical interface to create the VLAN on)
  - $2: vlan_id (description: The ID of the VLAN to create)
  - $3: mtu (description: The Maximum Transmission Unit size, defaults to 1500 if not provided)
- **outputs**: None
- **returns**: Always returns 0 to indicate success.
- **example usage**: n_vlan_create eth0 10 1500

7.0.767 Quality and security recommendations

1. Make sure you validate the inputs to this function to prevent any inadvertent modification of network configurations.
2. Be sure to handle error conditions appropriately. In the current implementation, errors are suppressed, particularly when attempting to delete an existing VLAN.
3. Always ensure that the VLAN ID and MTU values provided are within the acceptable range before creating the VLAN.
4. Make sure to sanitize the phys_iface input to prevent possible command injection attacks.

7.0.768 _osvc_kv_set

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

Function signature: e83bb88e314eb7098eb2ed8868cdecf150f3a7c4cec631f0ac14eadabbeaa0d3

7.0.769 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.

7.0.770 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"`

7.0.771 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.

7.0.772 package

Contained in `lib/node-functions.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: b9185dc00d625f6e970e86549720c3b815f20f10e6b152ea5faebf40f5dcd8f4

7.0.773 Function overview

The `package()` function is a simple utility function to copy over certain directories of a given starting directory to a package directory. It creates the package directory if it

does not exist and subsequently copies the `usr`, `etc` and `var` directories of the starting directory to the package directory.

7.0.774 Technical description

Name: `package`

Description: This function takes a source directory (startdir) and a destination directory (pkgdir), creates the destination directory if it does not exist, and copies the `usr`, `etc`, and `var` subdirectories from source to destination.

Globals:

- `pkgdir`: This is the name of the directory that the content will be copied to. It will be created if it does not exist.
- `startdir`: This is the directory where the `usr`, `etc`, and `var` directories exist that will be copied.

Arguments: The function does not take any arguments.

Outputs: The `usr`, `etc`, and `var` directories from `startdir` are copied over to `pkgdir`. In case these directories do not exist, the `cp` command would throw an error.

Returns: The function does not have explicit return statements, but if the operations complete successfully, it will implicitly return `0`, indicating success. Otherwise, it will return non-zero status codes indicating failure.

Example usage:

```
startdir=<path_to_source_directory>
pkgdir=<path_to_destination_directory>
package
```

7.0.775 Quality and security recommendations

1. Always use absolute paths for directories when copying data, especially in scripts used for packaging or deployment. Relative paths can potentially cause unwanted side effects.
2. Always check if the source directories exist before trying to copy them. If they do not, the script should exit with an error.
3. Add error handling to manage failure cases, at least by catching and logging any errors that occur during the copy process.
4. Be mindful of file permissions when copying directories. It might be necessary to ensure that permissions are kept intact or appropriately set in the destination. Remember that copy operations can potentially expose sensitive information.
5. For even greater reliability, consider adding a mechanism for verifying that the copy was successful (e.g., by checking that the files exist in the destination directory after the copy operation).

7.0.776 parse_apkindex_package

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

Function signature: c8272ffc02ed22252a554f027f250eccd763e445bdf8b52ad5784e2b6d338cbc

7.0.777 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.

7.0.778 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"
  ```

7.0.779 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.

7.0.780 `prepare_custom_repo_for_distro`

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

Function signature: f41fadc94c0ebb53ebb87776324ccfcc73903f665dfe9bfd0a9d3f4c9cbb827c

7.0.781 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.

7.0.782 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"`

7.0.783 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.

7.0.784 `print_cluster_variables`

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

Function signature: c5583d8fd4715e19ec442b98a50c60b43c9b20d0f5892f8d147bbb29263d00fa

7.0.785 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.

7.0.786 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`

7.0.787 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.

7.0.788 `_print_meta`

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

Function signature: 8f66cc2d9e01b400ddc05a00d6399036188baa213e119c36c4340e95320bcf7f

7.0.789 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.

7.0.790 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"`

7.0.791 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.

7.0.792 `register_source_file`

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

Function signature: 3a47ee217d4c21d4f884f60289c0b97418f2bea9352f1117fe8517c7bb30bb0b

7.0.793 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.

7.0.794 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"
```

7.0.795 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.

7.0.796 `remote_function_lib`

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

Function signature: b6d9cd335e4f61b186f614aa07279b2f8bdd77298bf573a231ffd21869c18118

7.0.797 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.

7.0.798 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.

7.0.799 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.

7.0.800 `remote_log`

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

Function signature: 1f586056ba1b573eed69d32639c3940c1c742ba73af4aae917a1d65d36c5367c

7.0.801 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.

7.0.802 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"
```

7.0.803 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.

7.0.804 `resolve_alpine_dependencies`

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

Function signature: 689d5ee3849151be4f6f15527c700df7d9c8156b9a6e4b876f2b82c3b22f9d9e

7.0.805 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.

7.0.806 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"`

7.0.807 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.

7.0.808 `rocky_latest_version`

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

Function signature: f42e139a07133ae36c4e9783c1fbb144e1167e59537bd374c0b346f853c77a3d

7.0.809 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.

7.0.810 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}"
```

7.0.811 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.

7.0.812 `script_render_template`

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

Function signature: 064ddcb49f3688c1b0ede8ce884eae36c9ac96f5117338bdce1f62d5c6960a67

7.0.813 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.

7.0.814 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'`.

7.0.815 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.

7.0.816 `select_cluster`

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

Function signature: d6887af36fa9b9a8372e021a7e7c69665b0d3eee270caacdddd223f760546494

7.0.817 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.

7.0.818 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`

7.0.819 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.

7.0.820 `select_network_interface`

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

Function signature: 019a6a43a2b588278bd5945964c48b92ffb331b67f1cacc18bedd6dd6828d661

7.0.821 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.

7.0.822 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")`

7.0.823 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?

7.0.824 `set_active_cluster`

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

Function signature: ff67ed1fe94af1bc0a6ebd9436efd66e00f5b5f9905b2979af037bdf49fce60e

7.0.825 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.

7.0.826 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
```

7.0.827 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.

7.0.828 `start_pre`

Contained in `lib/host-scripts.d/alpine.d/BUILD/10-build_opensvc.sh`

Function signature: 84093d24ed6e059f89dd6ad32c2ff11b16b0fd2540a9f3c88ab3932e8a93570b

7.0.829 Function Overview

The `start_pre` function in bash script is primarily used to handle out of memory situations and ensure that the essential data directory exists. Initially, it checks whether the current bash process has write permissions on `/proc/self/oom_score_adj` file to protect the process from being terminated due to out of memory (OOM) conditions. Afterwards, it validates that the data directory `/var/lib/opensvc` exists or else creates it.

7.0.830 Technical Description

Name: `start_pre`

Description: The function verifies if the existing process has the writing permission for the OOM score adjustment. If it does, it changes the value to -1000 therein, giving the process a lower likelihood of being killed during an OOM scenario. Following that, it checks for the existence of a directory, and if absent, it takes responsibility for creating it.

Globals: None

Arguments: None

Outputs: Writes `-1000` to `/proc/self/oom_score_adj` if it has write permissions. Creates `/var/lib/opensvc` directory if it doesn't already exist.

Returns: Nothing

Example Usage:

```
source script_name.sh
start_pre
```

7.0.831 Quality and Security Recommendations

1. It's highly recommended to add error handling for directory creation. If `mkdir` fails, it should notify the user.
2. The number `-1000` is hardcoded. It's advisable to make it a constant, with an understandable name, at the start of the script.

3. To further enhance security, consider modifying permissions of `/var/lib/opensvc` directory in the script, as per the use case requirements after creation.
4. Validate the impact of changing the OOM score beforehand, especially in production environments.
5. Log significant script actions and errors for debugging purposes.

7.0.832 `storage_deprovision_volume`

Contained in `lib/host-scripts.d/common.d/storage-management.sh`

Function signature: 668ca90d247ae9b85bd28558e102cecbf0dd119fbbf5fca3b61e894cf67082d7

7.0.833 Function overview

The `storage_deprovision_volume` function is primarily used to deprovision a given volume in storage. It first parses the arguments for IQN and volume name, then ensures that the host type is suitable for deprovisioning. Additionally, it verifies the existence of a local zpool name. The function subsequently deletes the iSCSI target and the volume based on the provided IQN and volume name. Failure in deletion results in an error log, and successful deprovisioning returns 0.

7.0.834 Technical description

- **Name**: `storage_deprovision_volume`
- **Description**: Parses arguments and deprovisions a specified storage volume by deleting the iSCSI target and the volume itself. Checks for host type and presence of a local zpool name.
- **Globals**: None.
- **Arguments**:
  - `$1`: Action flag. Possible flags are `--iqn` (sets internal iqn variable) and `--zvol-name` (sets internal zvol_name variable)
  - `$2`: Corresponding value for the flag set by `$1` (either iqn value or zvol name depending on `$1`).
- **Outputs**: Logs into a remote system information about the steps made by the function. If errors occur, they are reported in the log.
- **Returns**:
  - `1` if invalid flag is set, required flags are not set, host type is not 'SCH', zpool name could not be determined or deletion of zvol fails.
  - `0` if the volume was successfully deprovisioned.
- **Example usage**:

```
storage_deprovision_volume --iqn
 ↪  iqn.2003-01.org.linux-iscsi.localhost:x8664.sn.d33fadd1d40
 ↪  --zvol-name zvol1
```

7.0.835 Quality and security recommendations

1. Make sure only authorized users can run this function to ensure that volumes are not accidentally deprovisioned.
2. Enhance error checking to catch unknown flags and handle them appropriately.
3. Consider adding functionality to backup the volume before deletion to allow recovery in case of accidental deprovisioning.
4. Always use secure credentials when logging into the remote system to protect the integrity of the data.
5. Consider hardening the script by restricting the ability to deprovision based on additional factors, such as the time of day or the load on the system.
6. Regularly review and audit the logs produced by the function for any irregular activities.

7.0.836 `storage_get_available_space`

Contained in `lib/host-scripts.d/common.d/storage-management.sh`

Function signature: 0ecd662d000f3348b1348219cc1c7541ead05b93b29de5edfc244de079f38d03

7.0.837 Function overview

The function `storage_get_available_space` is a bash function used to find out the available space in a specific ZPOOL. The function uses ZFS commands to determine the amount of available space in the ZPOOL. It first fetches the ZPOOL name from a remote host and then checks if the ZPOOL name could be fetched successfully. It then queries ZFS for the available space in that ZPOOL and logs an error if this is not successful. Finally, it echos the available space in bytes and returns 0 on successful execution.

7.0.838 Technical description

- **Function name**: `storage_get_available_space`
- **Description**: This function retrieves the available storage space in a specified ZPOOL residing on a remote host. The ZPOOL name is obtained via the remote_host_variable function.
- **Globals**: [ZPOOL_NAME: the ZPOOL's name in the remote host]
- **Arguments**: None
- **Outputs**:
    1. Logs errors if the ZPOOL name cannot be determined, or if ZFS fails to retrieve available space.
    2. Prints the available space in bytes if the function executes successfully.
- **Returns**:
    1. If the ZPOOL name cannot be determined or ZFS fails to retrieve available space, the function returns 1.

2. Upon successful execution, the function returns `0`.

- **Example usage**: `storage_get_available_space`

7.0.839 Quality and Security Recommendations

1. Ensure that the remote_host_variable and remote_log functions are secure and cannot be exploited to execute remote shell commands.
2. Make sure that error logs do not reveal sensitive information about the system, which could be utilized by an attacker.
3. Validate the output of the zfs get command to ensure it can't be manipulated to inject malicious code.
4. Check the network connection between the local and remote systems securely to prevent MITM (Man in the Middle) attacks.
5. Consider adding more error handling and logs for network issues or ZFS failures.
6. Regularly update the ZFS command-line tools to benefit from the most recent security patches and improvements.

7.0.840 `storage_get_host_vlan`

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

Function signature: e63a3a1fba8ac0a209a0abe4bcc73383ffdfe3060c25018bef1faedec9aa1b77

7.0.841 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.

7.0.842 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"`

7.0.843 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.

7.0.844 `storage_parse_capacity`

Contained in `lib/host-scripts.d/common.d/storage-management.sh`

Function signature: c67957f8ee4be8442088f8824ef3828b5a9b2d67c4b4f352df2b050bf6c8bbee

7.0.845 Function overview

The Bash function `storage_parse_capacity()` takes a string argument indicating a data size (with an optional suffix denoting the scale e.g., K, M, G, or T for Kilobytes, Megabytes, Gigabytes, and Terabytes respectively). The function will parse this string and return the data size in bytes. The function will return 1 and terminate if the input does not match the expected format, i.e., a numerical value optionally followed by a suffix (K, M, G, or T).

7.0.846 Technical description

Definition block for `storage_parse_capacity()` function:

- **name**: `storage_parse_capacity()`
- **description**: Parses a string argument depicting a data size with an optional suffix (K, M, G, T) and returns the equivalent size in bytes.
- **globals**: None
- **arguments**:

  - `$1`: The capacity string to be parsed. Could be a plain number (considered as bytes), or suffixed with K, M, G, or T (case-insensitive) to denote Kilobytes, Megabytes, Gigabytes, and Terabytes, respectively.
- **outputs**: The function echoes the parsed capacity (data size) in bytes.
- **returns**:
  - 0: if the processing was successful.
  - 1: if the input string is empty or does not match the expected format.
- **example usage**: `$ storage_parse_capacity 20K` would output `20480`.

7.0.847 Quality and security recommendations

1. Add input validation: There should be some additional error handling to make sure the value before the suffix is a valid number. Currently, non-numeric characters before the suffix can lead to unexpected behaviour.
2. Use consistent error: The function should always echo an error message to stderr whenever it returns 1. This would make it easier for users to understand any error that the function encountered during its execution.
3. Create unit tests: To ensure that the function consistently works as expected, create some unit tests that will run the function with different inputs and compare the return values with expected results.

7.0.848 `storage_provision_volume`

Contained in `lib/host-scripts.d/common.d/storage-management.sh`

Function signature: 5e3861e2975c7a31100612e49933846099de90b2882d09056b15392c4698cf6b

7.0.849 Function Overview

The `storage_provision_volume()` function is designed to automate the process of provisioning a storage volume within a networked storage solution. The function accepts three parameters to define the fully qualified iSCSI Qualified Name (IQN), the storage capacity, and the zvol name of the volume to be created. The function then creates an iSCSI target, or storage resource, that other iSCSI initiators on the network can access.

7.0.850 Technical Description

storage_provision_volume()

- **Description:** Provisions a storage volume within a networked storage solution by creating an iSCSI target.
- **Globals:**
  - `host_type`: checks to verify this is a storage host.
  - `zpool`: gets local zpool name.

- **Arguments:**
  - `--iqn` (`$2`): The IQN of the iSCSI.
  - `--capacity` (`$2`): Storage size requirement in appropriate units (Byte, Kilobyte, Megabyte, or Gigabyte).
  - `--zvol-name` (`$2`): The name of the volume to be created.
- **Outputs:**
  - Validates required parameters.
  - Verifies that this is a storage host.
  - Calculates and reports available space, ensuring enough space exists for the requested volume.
- **Returns:**
  - `1` if an error occurs, such as missing required parameters, incorrect host type, running out of available space, or failure in creating zvol or iSCSI target.
  - `0` if the volume is successfully provisioned.
- **Example usage:**

```
storage_provision_volume --iqn
 ↪  iqn.2021-05.com.example:storage:disk1 --capacity 1G
 ↪  --zvol-name disk1
```

7.0.851 Quality and Security recommendations

1. The function should check the validity of the passed arguements which includes checking if the `iqn` and `zvol-name` are properly formatted and if the capacity is realistically feasible before attempting to provision the storage volume.
2. It may be beneficial to have an additional parameter to choose the type of volume to be provisioned—block, file or object—to add versatility to the function.
3. This function could further be improved by providing a rollback mechanism for partial completions, in case the storage provisioning operation fails midway.
4. Always ensure that error messages do not disclose too much information, which might end up being a security risk. For instance, revealing the host type or zpool name could provide useful information to malicious users.
5. To reduce the risk of injection vulnerabilities, ensure that all parameters within the shell command are appropriately escaped or quoted.
6. For a more robust implementation, consider using a try-catch mechanism to handle unexpected errors. This will prevent the script from crashing and provide a more elegant way of logging the error.

7.0.852 `strip_quotes`

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

Function signature: b2892c33de60887f65c7f2b7946fafee0d873041d3a244be57b49c6339bb1cb5

7.0.853 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.

7.0.854 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!'"`

7.0.855 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.

7.0.856 `_supervisor_append_once`

Contained in `lib/functions.d/configure-supervisor.sh`

Function signature: 5f79fe5e7a4a7d3eb591b9e371a140f9bd1de60abf5e87d0e47082bfff056191

7.0.857 Function Overview

The function `_supervisor_append_once` is designed to modify the settings for Supervisor, a control system for UNIX-based servers. It takes two arguments—`stanza` and `block`. The function checks the supervisor configuration file, which is determined by the environment variable `${SUPERVISORD_CONF}`. If the stated `stanza` is not present in the configuration file, it then appends the corresponding `block` at the end of the file. This function is used multiple times subsequently in the script to ensure Supervisor is configured to securely and efficiently manage various service programs.

7.0.858 Technical Description

- **Name**: `_supervisor_append_once`
- **Description**: This function appends configuration blocks to the Supervisor configuration file for a specified service program, but only if that block does not already exist. It is used in managing UNIX based servers.
- **Globals**: [`${SUPERVISORD_CONF}`: Path to the Supervisor configuration file]
- **Arguments**:
  - `$1`: `stanza`— the name of the service program, e.g. `program:nginx`
  - `$2`: `block`— the complete configuration string to be appended.
- **Outputs**: Appends a configuration block to the Supervisor configuration file for a specific service program.
- **Returns**: Nothing.
- **Example Usage**: `_supervisor_append_once "program:dnsmasq" "$(cat <<EOF`
  - This usage of the function appends dnsmasq related configurations to the Supervisor configuration file.

7.0.859 Quality and Security Recommendations

1. Enhance error handling to prevent the potential mishandling of non-existent files.
2. Introduce a mechanism to backup the original configuration file before making changes to it.
3. Expand on arguments validation by checking for empty or null arguments.
4. Since shell scripts are susceptible to injection attacks, consider potential sanitization steps for the `stanza` and `block` variables.
5. Avoid hard coding paths and consider converting them into arguments or environment variables to improve scalability.

7.0.860 `sync_alpine_packages`

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

Function signature: b2fb9504a3d2f06000545f9301edac370a296cec8ec2e779dd1d02496acfeafc

7.0.861 Function Overview

The function `sync_alpine_packages` is used to synchronize the Alpine software packages. It takes the version of Alpine, the version of the mirror, the name of the repository, and a set of package names as inputs. It locates the desired Alpine packages and their dependencies from the specified repository, downloads them, and finally places all the files at the desired destination. If any issues happen during the process such as failing to create directory, download or extract APKINDEX, or resolve dependencies, the function handles errors and cleans up temporary storage to maintain a safe state.

7.0.862 Technical Description

- **Name:** `sync_alpine_packages`
- **Description:** This function synchronizes software packages of given package names from specified Alpine Linux mirror and repository.
- **Globals:**
  - `HPS_DISTROS_DIR`: The base directory for the distros.
- **Arguments:**
  - `$1`: Alpine version.
  - `$2`: Mirror version.
  - `$3`: Repo name.
  - Rest arguments: Package names.
- **Outputs:** Downloaded packages at `${HPS_DISTROS_DIR}/alpine-${alpine_version}/apks/${rep`
- **Returns:**
  - 0: On successful completion.
  - 2: If any error occurred during package synchronization.
- **Example Usage:** `sync_alpine_packages 3.12 3.12 main bash curl`

7.0.863 Quality and Security Recommendations

1. Always check if required arguments are passed before executing the function.
2. All variables inside the function should be local to avoid potential naming collisions.
3. Validate user inputs to prevent potential command injection attacks.
4. Handle all error paths gracefully, such as download failures or file system limitations.
5. Use more secure methods to download packages, if available, such as HTTPS instead of HTTP.
6. Consider verifying packages' authenticity and ensuring data integrity through checksums or digital signatures.
7. Document return codes of a function and their meanings for easier debugging and maintenance.
8. Cleaning up temporary storage areas is a good practice but also consider a more robust cleanup mechanism in case of unexpected termination.

7.0.864 sync_alpine_repo_arch

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

Function signature: 1e50c32388b6c1ff2be449bd1f28c57fd1d94b6fd34b27c243dba0e74d19f4e4

7.0.865 Function Overview

The function sync_alpine_repo_arch is designed for synchronizing alpine repositories to a local directory. It uses parameters such as alpine version, mirror version, repository name, and architecture to determine the repository to sync. The destination directory is built using the environmental variable and provided parameters, defaulting to x86_64 architecture if no other is provided. This function includes numerous error checks such as ensuring the destination directory can be created, and the available disk space is sufficient for downloads.

7.0.866 Technical description

- **Name**: sync_alpine_repo_arch
- **Description**: This function synchronizes an Alpine Linux package repository to a local directory. It retrieves the file list from the repository, checks if there's enough disk space, then uses wget to download the packages.
- **Globals**: [HPS_DISTROS_DIR: The base directory for syncing distros]
- **Arguments**:
  - [$1: Alpine Linux version (e.g., v3.12)]
  - [$2: Version of the mirror (e.g., edge)]
  - [$3: Name of the package repository (e.g., main)]
  - [$4: Architecture (e.g., x86_64), optional, default x86_64]
- **Outputs**: Logs info, debug, and error messages. Stores downloaded files in the dest_dir defined in the function.
- **Returns**: 2 if an error occurred that prevented successful function execution, return value of the validate_apkindex function otherwise.
- **Example usage**: sync_alpine_repo_arch v3.12 edge main x86_64

7.0.867 Quality and Security Recommendations

1. Enhance error handling: Presently, the function returns 2 when an error occurs, thus doesn't differentiate between failures. It would be beneficial to have unique error codes for the different failure points.
2. Download verification: After each download, consider adding functionality that verifies the authenticity and integrity of the downloaded files. This not only helps in ensuring that the complete file has been downloaded but also safeguards against any potential security concerns.
3. User input validation: Validate user inputs earlier in the function to prevent possible misuse, such as directory traversal.

4. Secure Temporary file handling: Ensure the security of temporary files by setting appropriate permissions and minimizing their lifespan.
5. Implement error retries: Temporary issues may hamper the download process, therefore implementing reparative measures such as retries with backoff periods could increase reliability.
6. Insert proper cleanup actions: Even if the function fails at any point, it should clean up any temporary files created during its execution. Implementing a trap for `EXIT` signal could be beneficial for this task.

7.0.868 `tch_apkovol_create`

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

Function signature: ed1d82a379d9d6afbfe07af535414a6e06f0d874630568ae59ea3561918ad0ec

7.0.869 Function Overview

`tch_apkovol_create` is a Bash function primarily responsible for collecting configuration details (such as the IP of the gateway and the latest Alpine version) and generating the Alpine apkovl (Alpine Linux package overlay) tarball in the specified output file. The function also handles temporary directory creation for building apkovl components and cleanup actions once the tarball is created. It also manages logging of the operation statuses.

7.0.870 Technical Description

- **Name:** `tch_apkovol_create`
- **Description:** Used for collecting configuration details and creating an Alpine apkovl tarball containing essential configuration files and scripts for a target system.
- **Globals:** None
- **Arguments:**
  - `$1: output_file` – The file path at which to generate the apkovl tarball
- **Outputs:** Logs informational, error and debug messages related to the progress and status of the Alpine apkovl creation process.
- **Returns:**
  - `0` if the apkovl creation process is successful
  - `1` if an error is encountered during the process
- **Example Usage:**

```
tch_apkovol_create "/path/to/output/file"
```

7.0.871 Quality and Security Recommendations

1. Always validate function inputs for proper format and expected data type to prevent unexpected behavior or errors.
2. Use a more unique naming scheme for temporary directories to reduce the risk of naming collisions.
3. Implement more granular error handling for the different stages of the apkovl creation process, ensuring the cleanup of any created resources.
4. Consider better logging for debug, informational and error messages across the function for easier troubleshooting and tracing.
5. Handle exceptions that might occur when querying configuration settings.
6. Secure the creation, utilization and deletion of the temporary directory.

7.0.872 `ui_clear_screen`

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

Function signature: 0493ca3490c6e95475fb08d2f387298f827857616ec67718e0dd7bc3268a31d2

7.0.873 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.

7.0.874 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()`

7.0.875 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.

7.0.876 `__ui_log`

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

Function signature: d3d49681e2b691a5ff908ab2cfc80feb43c6f2c7f2aa6c60521377957fc9244b

7.0.877 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.

7.0.878 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"
```

7.0.879 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.

7.0.880 `ui_menu_select`

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

Function signature: 228d39d076d4308652684c61dd46d71781022466ed70155a37a899acdc69da71

7.0.881 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.

7.0.882 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[@]}"
```

7.0.883 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.

7.0.884 `ui_pause`

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

Function signature: 1636563cd29eae7fb011743bbb84e1bd4b67567168166cfc3cd0cf46472383ec

7.0.885 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.

7.0.886 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].
```

7.0.887 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.

7.0.888 `ui_print_header`

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

Function signature: 57e2fede290b133fd0e31c859a63c119ad15a0eea0326adcc61d2c65983dfecc

7.0.889 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.

7.0.890 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
==================================
```

7.0.891 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.

7.0.892 `ui_prompt_text`

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

Function signature: 698fb0f6a52fc5fb7d346cb2755ab85d4e44cef66d1ac20f8f40870457b2970a

7.0.893 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.

7.0.894 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"`

7.0.895 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.

7.0.896 `ui_prompt_yesno`

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

Function signature: 5b56e1af7169ad6721f24f1f595cee61d779f0b22963936d53073e284b177e9c

7.0.897 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.

7.0.898 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
```

7.0.899 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.

7.0.900 `unmount_distro_iso`

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

Function signature: 8736c022aa1702dffb2efe53cc382894bbe29e4a44b302d6fe6c2f67439871c6

7.0.901 Function Overview

The function `unmount_distro_iso` is used to unmount a distribution ISO. Given the string name of a distribution, it attempts to unmount the ISO if it is currently mounted. The ISOs are stored in the directory specified by the global variable `HPS_DISTROS_DIR`. If the ISO is not mounted, it returns 0, otherwise it logs the attempts to unmount the ISO and returns 0 if it is successfully unmounted and 1 otherwise.

7.0.902 Technical Description

- **name**: unmount_distro_iso
- **description**: Attempts to unmount a given distribution ISO.
- **globals**:
  - HPS_DISTROS_DIR: The directory of where the ISO’s are stored.
- **arguments**:
  - $1: string name of the distribution (DISTRO_STRING)
  - $2: none
- **outputs**: Logs info level messages documenting the unmounting attempt.
- **returns**:
  - Returns 0 if the ISO is not mounted or successful in unmounting.
  - Returns 1 if it fails to unmount the ISO.
- **example usage**: unmount_distro_iso ubuntu

7.0.903 Quality and Security Recommendations

1. Add error checking for the input arguments. Currently the function assumes that the input will be correct and does not check if the distribution string is empty or if it contains illegal characters.
2. Improve logging. On failure, consider logging the error message output from the `mount` command.
3. Use absolute paths. Relying on relative paths can lead to unexpected behaviour, depending on the working directory when the script is executed.
4. Check that `HPS_DISTROS_DIR` value is set before using it to avoid errors.
5. Make sure that appropriate permissions and ownership settings are in place. This can prevent unauthorized access or modifications.

7.0.904 `update_distro_iso`

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

Function signature: 2cc9510e14e685d9bd46ee12b37cf92d19a872fbfd40ed5f9884b79c76dc02d6

7.0.905 Function overview

The `update_distro_iso` function is a Bash function used to update an ISO image of a given Linux distribution. This function takes a single string argument `DISTRO_STRING` representing the distribution to update. It then unmounts the respective distribution ISO, prompts users to update the ISO file manually, checks if the ISO file exists and is properly updated, and finally re-mounts the ISO. If any step fails, the function returns with an exit code of 1.

7.0.906 Technical description

- **Name:** `update_distro_iso`
- **Description:** This bash function is designed to update the ISO image of a desired Linux distribution. It unmounts the currently mounted ISO, prompts user to manually update the ISO file, verifies if the updated ISO file exists, and remounts the updated ISO. If any of these steps fails, it outputs an error message and exits with a 1 status.
- **Globals:** [`HPS_DISTROS_DIR`: Directory path where the Linux distributions ISO are stored]
- **Arguments:** [`$1: DISTRO_STRING`: String representing the Linux distribution to be updated]
- **Outputs:** Messages detailing the status and results of each operation, including potential errors.
- **Returns:** The function returns 1 if any step of the process fails. If everything proceeds smoothly, there is no explicit return statement, so the function's status is the exit status of the last command executed (per bash's standard behavior).

- **Example usage:** `shell update_distro_iso "x86-Ubuntu-16.04"`

7.0.907 Quality and security recommendations

1. Validate that `DISTRO_STRING` is not empty at the beginning of the function to avoid unnecessary operations.
2. Consider checking that the manual ISO update was successful after the `Press ENTER when ready to re-mount...` statement.
3. Implement file existence and readability checks before trying to unmount or mount ISO files to avoid potential errors.
4. Consider using explicit return and exit codes to help clarify the function's behavior during debugging.
5. It's a good practice to use proper sanitization to free shell commands from potential code injection attacks. Ensure that user inputs are sanitized and safe before constructing paths based on them.

7.0.908 `update_dns_dhcp_files`

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

Function signature: 03499dd691020e6c8242d0a4a804e259ee89585ebf7a058cf6c54eefbd9da8f7

7.0.909 Function Overview

The Bash function `update_dns_dhcp_files` is designed to update DNS (Domain Name System) and DHCP (Dynamic Host Configuration Protocol) configuration files in a network environment. It does this by utilizing other internally defined functions, logging information, and error messages to keep track of the process. If either of the file update operations (DNS or DHCP) fails, the function will return an error. If both operations succeed, the dnsmasq service is reloaded to apply the new configurations, and the function successfully exits.

7.0.910 Technical Description

- **Name**: `update_dns_dhcp_files`
- **Description**: This bash function updates DNS and DHCP configuration files. The update is considered successful when the DHCP addresses file and DNS hosts file both build without errors. If either file fails to build, the function logs an error message and a failure status is returned. If both files build successfully, `dnsmasq` is reloaded to pick up the new configuration.
- **Globals**: None.
- **Arguments**: None.
- **Outputs**: Logs [INFO] and [ERROR] messages using the inbuilt `hps_log` function.
- **Returns**:

  - `0` If updating both DNS and DHCP files is successful.
  - `1` If building either DNS hosts file or DHCP addresses file fails.
- **Example usage**: The function is used without any arguments i.e

```
update_dns_dhcp_files
```

7.0.911 Quality and Security Recommendations

1. Consider using more detailed and unique error messages to assist debugging process and increase maintainability.
2. Techniques such as input validation and data sanitization should be implemented to increase security.
3. Always prefer using local variables inside a function to avoid side effects and accidental modification of global variables.
4. For long running processes, consider using methods to keep the user informed of the progress instead of simply running in the background.
5. Implement logging levels in `hps_log` function to control the verbosity of the logs in different environments (production, staging, development etc).
6. It might be helpful to wrap the dnsmasq service reload in a try-catch block to handle unexpected errors with the service restart.

7.0.912 `url_decode`

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

Function signature: ed859e291b1b9e1c8fb1a90d6106ac8f5001b644c3c5f7c6894fe11146c43e68

7.0.913 Function overview

The Bash function `url_decode` receives a string with URL-encoded format as an argument, and translates it to a plain string. After that, the function prepares additional information (origin identifier, client type), finishes the message and assigns it to `msg`. This message is then sent to syslog and written to a logfile if possible. If it's not possible to write to logfile, an error message is sent to syslog.

7.0.914 Technical description

- **Name**: `url_decode()`
- **Description**: This function takes a string in URL-encoded format, decodes it and prepares it to be sent to syslog or written to a log file.
- **Globals**: None
- **Arguments**: `$1`- URL encoded string that needs to be decoded.
- **Outputs**: Sends a message to syslog and writes it to a log file. If the log file is not writable, sends an error message to syslog.
- **Returns**: None

- **Example usage**:

```
url_decode "Hello%20World" # with input "Hello%20World" compatible
 ↪  with the URL-encoded format, outputs "Hello World"
```

7.0.915 Quality and security recommendations

1. Always ensure the input passed to the `url_decode` function is safe and free from any malicious scripts or injections.
2. Verify that the log file location specified is secure and the log data does not leak any sensitive information.
3. It is always a good security practice to not run scripts with elevated permissions unless necessary. Always check the permission level required by your script and manage it accordingly.
4. Handle the errors effectively within the function to ensure no crash occurs if the function receives unexpected input formats or if there are issues with the syslog service or file permissions.
5. Perform regular audits and monitor the system logs to detect any abnormal behavior or unauthorized access attempts in the system.

7.0.916 `urlencode`

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

Function signature: f758d39e7a343eef82fc4e92ae1358118cf9a79d8accf9f5013313b5448282ac

7.0.917 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.

7.0.918 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
```

7.0.919 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.

7.0.920 `validate_alpine_repository`

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

Function signature: 40cdbdfa69d128c0e53275f522f4913827561f596df427e85b5d208433153364

7.0.921 Function overview

The `validate_alpine_repository` function is used to validate alpine repositories set on a given directory (defined by HPS_DISTROS_DIR). It first checks if the environment variable HPS_DISTROS_DIR is set; if not, an error is logged and the function returns. If the Alpine version is not provided as an argument, it attempts to auto-detect one. Next, it builds the repository directory based on provided or auto-detected Alpine version and HPS_DISTROS_DIR. It then validates the existence of the repository directory and its APKINDEX file. Finally, it checks if the number of packages in the repository exceeds a defined minimum expectation.

7.0.922 Technical description

- Name: `validate_alpine_repository`
- Description: Validates a set Alpine repository.
- Globals: [HPS_DISTROS_DIR: Location of the Alpine distribution directories]

- Arguments: [$1: Alpine version. If not provided, get_latest_alpine_version is used to find it, $2: name of repository, defaults to main if not provided]
- Outputs: Logs detailing the validation process and validation outcome.
- Returns: 0 if validation is successful; 1 otherwise.
- Example usage: validate_alpine_repository 3.9 main

7.0.923 Quality and security recommendations

1. Enforce strict argument checking to ensure all necessary inputs are provided, and they are in the correct format.
2. Implement a feature that handles unexpected errors or exceptions to prevent potential security risks or system crashes.
3. Add more detailed logging for each step in the function to facilitate easier debugging and maintenance.
4. Consider using more secure methods for file and directory checking to avoid potential security vulnerabilities associated with file traversals.
5. Protect the function against possible command injection attacks by validating and sanitizing input arguments.
6. Use more robust error handling techniques to provide useful feedback when the function encounters any errors during execution.
7. Secure the repositories with digital signatures or hashes to ensure their authenticity and integrity.

7.0.924 validate_apkindex

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

Function signature: 8b01a88e6110ddb2342de0b3ad1c7ac2c81b45e6b739b8d49cb390eeb12b97d9

7.0.925 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.

7.0.926 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 t
3. Where possible, avoid using global variables to avoid potential conflicts and incre
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
```

7.0.927 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.

7.0.928 `verify_checksum_signature`

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

Function signature: 6927fb684cf8e6a1fa690734435cbce8b702cdeef1affb7a199413ba1ed2e406

7.0.929 Function Overview

The `verify_checksum_signature` function takes four positional arguments to define the type of ISO file to check and the location from where to fetch the ISO's CHECKSUM and GPG key. This function is designed to verify the integrity and authenticity of a downloaded ISO file according to the checksum and GPG signature fetched from the file's distribution server.

Currently, the function only supports the Rocky Linux distribution.

7.0.930 Technical Description

- **Name:** `verify_checksum_signature`
- **Description:** This function verifies a specified ISO checksum and GPG signature to ensure the authenticity and integrity of the file.
- **Globals:**
  - `HPS_DISTROS_DIR` (Defaults to `/srv/hps-resources/distros`): Specifies the directory of the distros.
- **Arguments:**
  - `$1 (cpu)`: Specifies the CPU architecture.
  - `$2 (mfr)`: Specifies the manufacturer.
  - `$3 (osname)`: Specifies the operating system name.
  - `$4 (osver)`: Specifies the operating system version.
- **Outputs:** Various status updates printed to the console. Error messages are redirected to standard error.
- **Returns:** `0` on success, `1` if there is an error (such as the ISO not being found, a failed download, checksum mismatch, or signature verification failure).
- **Example Usage:** `bash verify_checksum_signature "x86_64" "rocky" "rockylinux" "8"`

7.0.931 Quality and Security Recommendations

1. The function could benefit from input validation to ensure the provided `cpu`, `mfr`, `osname`, and `osver` arguments are in the expected formats before they are concatenated into URLs.
2. The function may need to be modified to handle ISO files for operating systems other than Rocky Linux.
3. Consider the implementation of stronger error handling rather than simply returning `1` on an error. Detailed and distinct exit codes may help to better identify specific issues that could occur.
4. The echoing of status updates could be optionally silenced for running the script in a quiet mode.
5. Temporary directories and files should be securely deleted to prevent sensitive information from being exposed on the server.
6. Implement HTTPS download error handling to improve security and stability. This could be accomplished within the `curl` command statements.

7.0.932 `verify_required_repo_packages`

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

Function signature: a16392408f8b201975834d3d38029e3859302d45f825b35156dca0ae85e71609

7.0.933 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.

7.0.934 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`

7.0.935 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.

7.0.936 `verify_rocky_checksum_signature`

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

Function signature: 63d87412289beb590b23dead3edfbcd2afb85d4ee141526d80e8ad9104dcfbcb

7.0.937 Function overview

The `verify_rocky_checksum_signature` is a Bash function designed for verifying the checksum signature of a specified Rocky Linux ISO image. It accomplishes this by first downloading the checksum and signature associated with the provided version of the Rocky Linux ISO image, then importing the GPG key from the Rocky Linux server, and finally, verifying the GPG signature. If all passes, the function will then calculate and compare the sha256 checksum of a specified ISO file to ensure the ISO image has not been tampered with.

7.0.938 Technical description

- Name: `verify_rocky_checksum_signature`
- Description: This function downloads and verifies the checksum and its signature of a Rocky Linux ISO image. Additionally, it validates the SHA256 hash of the ISO file against the downloaded checksum.
- Globals:
  - `HPS_DISTROS_DIR`: A directory path where Rocky Linux distribution files are stored.
- Arguments:
  - `$1 (version)`: The version of Rocky Linux ISO image to verify.
- Outputs: The function will output various status messages indicating the status of download, GPG key import, and verification steps.
- Returns:
  - `0`: Success. The checksum signature has been verified and matches the checksum of the specified ISO image.
  - `1`: Failure. The GPG key import process was unsuccessful.
  - `2`: Failure. The checksum signature verification failed.
  - `3`: Failure. No checksum was found for the specified ISO image.
  - `4`: Failure. The checksum of the ISO file does not match the downloaded checksum.
- Example Usage:
  - `verify_rocky_checksum_signature "8.4"`

7.0.939 Quality and security recommendations

1. Always ensure to use secure and up-to-date versions of all tools and packages used in the function.
2. Instead of hardcoding the architecture (`x86_64`), consider making it an argument or an environment variable that is configurable by the user.
3. Consider checking whether the `curl` and `gpg` commands succeed immediately rather than using a mixture of exit status checks and error redirections.
4. Employ better error handling and provide more specific output messages in case of failures.
5. Avoid reassigning constants in the middle of the function, such as `checksum_path` and `sig_path`.
6. Uncomment the downloading commands for CHECKSUM and CHECKSUM.sig or explain why they are commented out.

7.0.940 `write_cluster_config`

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

Function signature: cf6f48116ee4f6ae2fa075ed74a4476f6c22ffe594564ebf704d40d3dce09039

7.0.941 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.

7.0.942 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"
```

7.0.943 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.

7.0.944 `zfs_get_defaults`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: 7075829ac859fa1a3d095289b6f87eeae5bfd9df08c9d1cfd66df7de132cf128

7.0.945 Function Overview

The function `zfs_get_defaults` is designed for setting sensible defaults for pool options (_POOL_OPTS) and ZFS properties (_ZFS_PROPS). This function helps to customize and optimize your ZFS filesystem according to your needs. Pool options include sector size, while ZFS properties cover settings such as compression type, access time, extended attribute style, Access Control List (ACL) type, mode, inheritance, node size, and log bias.

7.0.946 Technical Description

- Name: `zfs_get_defaults`
- Description: This function sets default values for pool options (_POOL_OPTS) and ZFS properties (_ZFS_PROPS). The pool options are safest for SSD/NVMe/HDD with 4K-sectors. The ZFS properties include features like compression, access time, ACLs, and others.
- Globals:
  - _POOL_OPTS: An array to store pool options.
  - _ZFS_PROPS: An array to store ZFS properties.
- Arguments:
  - $1: A reference to a variable intended to hold pool options.
  - $2: A reference to a variable intended to hold ZFS properties.
- Outputs: There are no explicit outputs besides the modified _POOL_OPTS and _ZFS_PROPS variables.
- Returns: This function does not have any explicit return values and does not generate exit status.
- Example usage: `zfs_get_defaults POOL_OPTS ZFS_PROPS`

7.0.947 Quality And Security Recommendations

1. Consider making the function's name more descriptive, such as `set_zfs_defaults`, to specify the action that the function performs.
2. For improved security, validate the variables that are passed into the function to ensure they allow item assignment.
3. When setting sensible defaults, remember to base it on the actual use case scenario and the nature of the data handled.
4. To enhance transparency and ease of debugging, consider implementing a logging system to record any changes made by the function.
5. The comments flagged with 'TODO' should be addressed. Consider implementing the -O props within the function as these can provide an additional level of customization for the user.

6. Always include error checking in your functions to catch unexpected scenarios and improve robustness.

7.0.948 zpool_create_on_free_disk

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: ff0bf00dc7c2ed8816d8a88a6b148a5993ba6e4c5d1ae70415c5960612576a80

7.0.949 1. Function overview

The `zpool_create_on_free_disk` function is a Bash utility for working with data storage in a Unix-like operating system. It initializes a zpool on free disk space using a specified strategy. It has preset values for variables such as `strategy` (defaults to "first"), `mpoint` (defaults to "/srv/storage"), `force`, `dry_run`, and `apply_default` all of which can be adjusted according to user requirements.

7.0.950 2. Technical description

- **Name**: zpool_create_on_free_disk
- **Description**: A bash function designed to initialize a zpool on available disk space. This function allows users to set their preference with a number of preset options to customize the setup according to their needs. The possible options include specifying a storage strategy, the mount point for the storage, and whether to force the operation, do a dry run or apply default settings.
- **Globals**:
  - `strategy`: Defines the strategy for initializing the zpool storage. Default is "first".
  - `mpoint`: The directory in which the initialized storage will be mounted. Default is "/srv/storage".
  - `force`: Defines whether or not to force the initialization. Default is 0 (do not force).
  - `dry_run`: Defines whether or not to perform a dry run. Default is 0 (do not dry run).
  - `apply_defaults`: Defines whether to apply the default settings. Default is 1 (apply defaults).
- **Arguments**:
  - `$1`: The first positional parameter is not explicitly used in this function.
  - `$2`: The second positional parameter is not explicitly used in this function.
- **Outputs**: Outputs the status of the zpool creation process.
- **Returns**: Returns a status code indicating the success or failure of creation.
- **Example usage**: To use this function, you would typically include in a Bash script like this:

```
zpool_create_on_free_disk
```

7.0.951 3. Quality and security recommendations

1. Validate input: Although this function does not take arguments, it is always good practice to ensure any input or sources from which input is derived are valid.
2. Error handling: Include error handling mechanisms for situations where the disk space is not available or the formatted storage cannot be mounted at the specified mount point.
3. Security: Ensure that the storage's mount point has appropriate permissions set, and sensitive data is securely managed.
4. Code clarity: Some variables are initialized but not used, these could be removed for improved code clarity.
5. Logging: Add comprehensive logging for tracking the sequence of operations. Logs would aid in debugging and resolving any issues that might arise.
6. Code Comments: Adding comments to explain the purpose of complex code blocks would make the function more maintainable.

7.0.952 `zpool_name_generate`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: 5995908b1c71b7b0931db1a09cf94c2257d6d0ed783b7e45ca8926f62b975cf6

7.0.953 Function Overview

The `zpool_name_generate` function is a Bash function that generates a ZFS pool (zpool) name based on certain parameters. It specifically takes an input, "class," and produces a zpool name incorporating information about the type of storage (like NVMe, SSD, HDD, ARC, or MIX), the cluster name, current Unix timestamp, and a random 3-byte hexadecimal identifier.

7.0.954 Technical Description

- **Name**: `zpool_name_generate`
- **Description**: This function receives a "class" parameter, which represents the type of storage, and generates a unique zpool name that includes the storage type, cluster name, Unix timestamp, and a random hexadecimal identifier.
- **Globals**: None
- **Arguments**:
  - `$1 (class)`: an indicator of the storage class. It could be either of `nvme`, `ssd`, `hdd`, `arc`, `mix`. If it's not set or doesn't match these, the function will return an error.
- **Outputs**: This function outputs a zpool name to stdout.

- **Returns**:
  - Return code 2 if the class argument is not given or doesn't match the expected values.
  - Return code of the `zpool_slug` function call if it fails.
- **Example Usage**:

```
$ zpool_name_generate nvme
```

7.0.955 Quality and Security Recommendations

1. The function doesn't validate the cluster name from the `remote_cluster_variable` function. Such validation could be beneficial to avoid creating zpools with incorrect or malicious names.
2. The random three-byte value is generated using both `/dev/urandom` and the `RANDOM` variable. To maintain consistency, consider using only one of these methods. Using `/dev/urandom` would make it more random and secure.
3. Consider adding error checking for critical operations like od and `printf`, which could fail due to various reasons.

7.0.956 `zpool_slug`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: 18bbbffc19d9426de745e5b45fe837f9d50ec9443122924bb19421975f877cbc

7.0.957 Function overview

The `zpool_slug()` function is designed to convert a given string into a slug, suitable for use in URL paths or IDs. This function takes up to two arguments: the string to convert and (optionally) the maximum length of the slug. It converts the string to lower case, replaces any non-alphanumeric characters with a hyphen and removes any consecutive or trailing hyphens. It then truncates the slug at the specified maximum length or at 12 characters if no length was provided.

7.0.958 Technical description

- **name**: `zpool_slug()`
- **description**: Converts a string into a slug of a specified or default length.
- **globals**: None.
- **arguments**: [`$1`: The string to be transformed into a slug, `$2`: Maximum length of the slug. This argument is optional, with a default length of 12 if left unspecified.]
- **outputs**: Prints the slug to stdout
- **returns**: None.

- **example usage**: `zpool_slug "Example String" 10` will output `example-str`

7.0.959 Quality and security recommendations

1. Right now, there is no explicit handling of invalid input, such as non-string values. Adding type checking and error handling for these scenarios would improve robustness.
2. Consider adding input sanitization to prevent any potential security issues (although current implementation is already reasonably safe due to removal of all non-alphanumeric characters).
3. Ensure that maximum slug length is not overly restrictive and take possible unicode characters into account.
4. Write unit tests for this function to guarantee it behaves as expected and validate the slug creation logic.
5. Specify locale in the script to ensure consistent character conversion, as the current implementation uses the locale setting of the running environment which can lead to unexpected results.

8 Deploying the disaster recovery node

Installing and configuring the DR node, synchronising data, and testing recovery procedures.

8.1 Deployment process

Stub: Steps for installing and integrating the DR node into the cluster.

8.2 Purpose of the DR node

Stub: Explain the role of the DR node in ensuring service continuity.

8.3 Failover and recovery testing

Stub: Procedures for verifying DR readiness and simulating failover scenarios.

8.4 Preparation

Stub: Hardware, storage, and network prerequisites for the DR node.

8.5 Synchronisation

Stub: Methods for keeping DR node data in sync with the primary environment.

9 Design Decisions

9.1 Choice of boot firmware: UEFI vs legacy BIOS for diskless servers

Decision **Legacy BIOS** selected as the boot firmware mode for diskless servers.

9.1.1 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.

9.1.2 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.

9.1.3 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.

9.1.4 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.

9.2 Choice of File System for iSCSI Export: ZFS vs. Btrfs

Decision ZFS selected as the primary file system for our iSCSI export volumes.

9.2.1 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.

9.2.2 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.

9.2.3 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.

9.2.4 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.

9.3 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.

9.3.1 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.

9.3.2 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.

9.3.3 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.

9.3.4 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.

10 HPS Network Topology Design

10.1 Objectives and Intent

10.1.1 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.

10.1.2 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)

10.2 Network Architecture Overview

10.2.1 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                        │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```

10.2.2 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.

10.2.3 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                    │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```

10.3 Bootstrap Process

10.3.1 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.

10.3.2 HPS Solution: Rapid VLAN Transition

```
Phase 1: Initial PXE (untagged, ~3-7 seconds)
```

```
┌─────────────────────────────────────────────┐
│ 1. Host NIC firmware initiates PXE          │
│ 2. DHCP request (untagged)                  │
│ 3. IPS responds: IP + iPXE bootloader       │
│ 4. Download iPXE binary (~100KB)            │
│ Duration: ~3-7 seconds total                │
└─────────────────────────────────────────────┘

         ↓
Phase 2: iPXE VLAN Configuration (~1-2 seconds)

┌─────────────────────────────────────────────┐
│ 5. iPXE executes embedded script:           │
│    - vcreate --tag 10 net0                  │
│    - dhcp net0-10                           │
│    - chain to boot_manager on VLAN 10       │
└─────────────────────────────────────────────┘

         ↓
Phase 3: Full Boot and Runtime (VLAN 10)

┌─────────────────────────────────────────────┐
│ 6. Download kernel, initrd from VLAN 10     │
│ 7. Boot OS with VLANs configured            │
│ 8. All provisioning on VLAN 10              │
│ 9. Runtime: VLAN 10, 20, 31, 32 active      │
└─────────────────────────────────────────────┘
```

Untagged exposure: ~3-7 seconds

All runtime operations: VLAN-tagged

10.3.3 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.

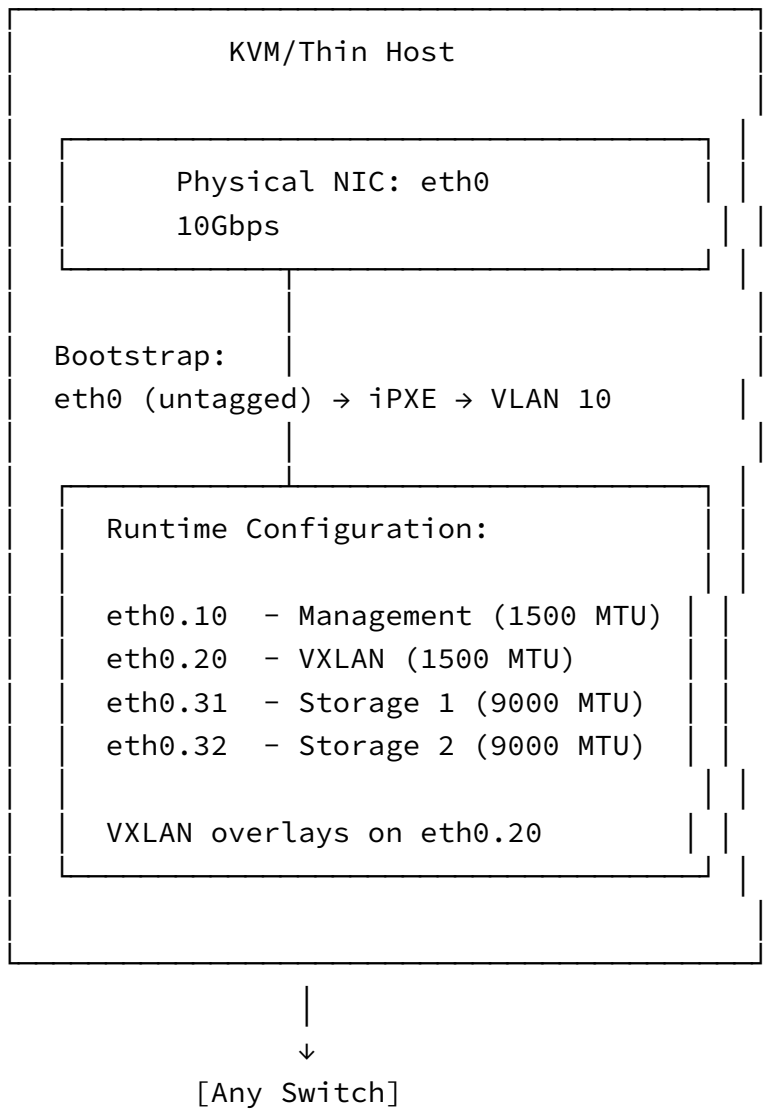
High Security Mode (Optional) No untagged bootstrap allowed. Requires: Pre-configured switch with VLAN 10 as native VLAN or out-of-band management (IPMI) for initial configuration. Suitable for: Highly regulated environments.

10.4 Deployment Stages

10.4.1 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:

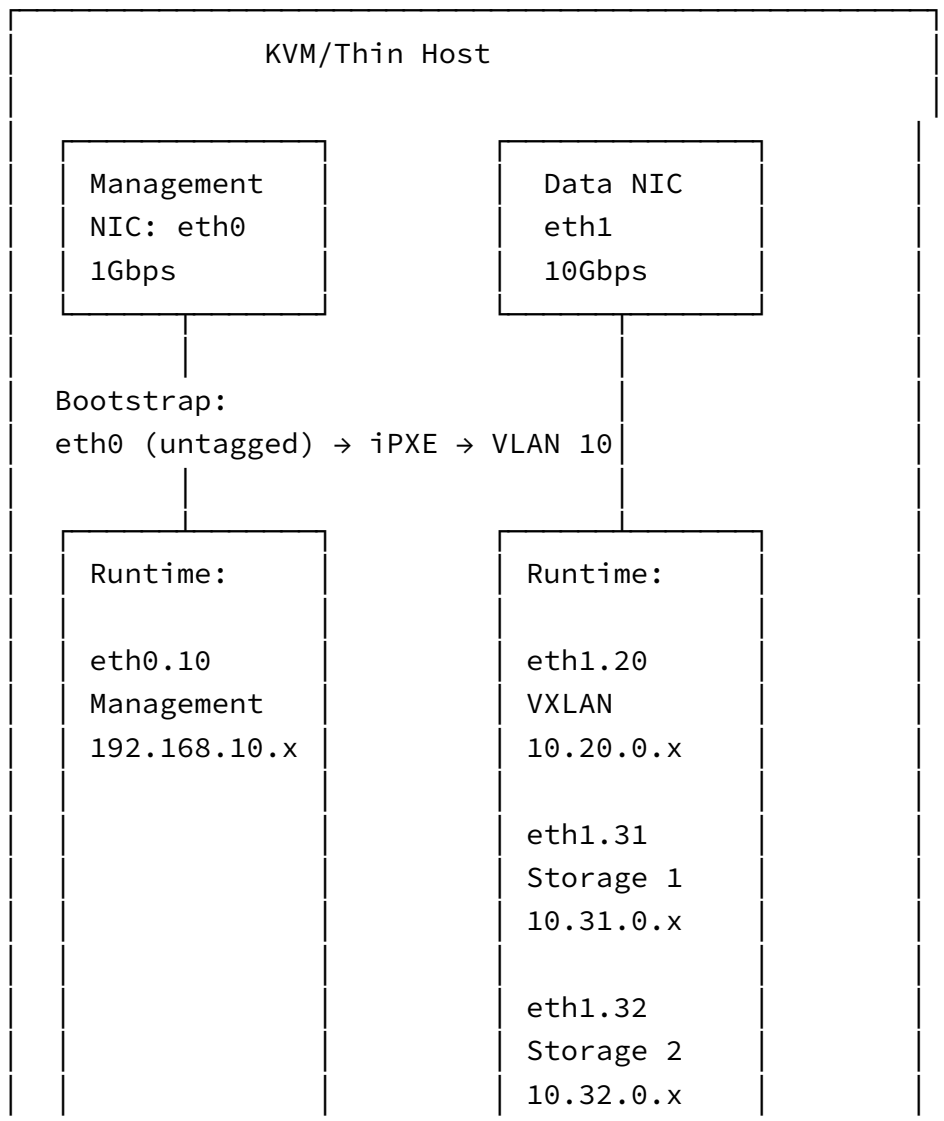
1. PXE boot (untagged) → iPXE
2. iPXE creates VLAN 10, gets IP
3. Downloads OS from VLAN 10
4. OS configures all VLANs (10, 20, 31, 32)

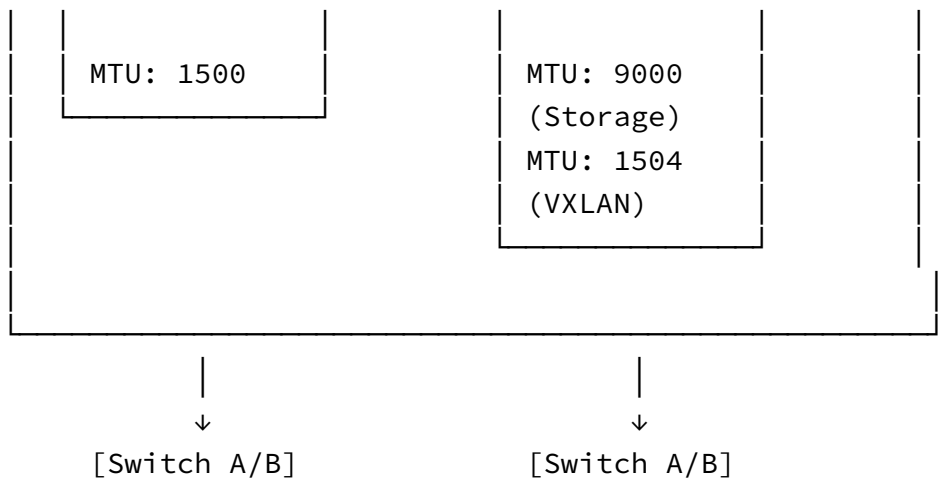
10.4.2 Profile 2: Dual NIC (Production Entry)

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:

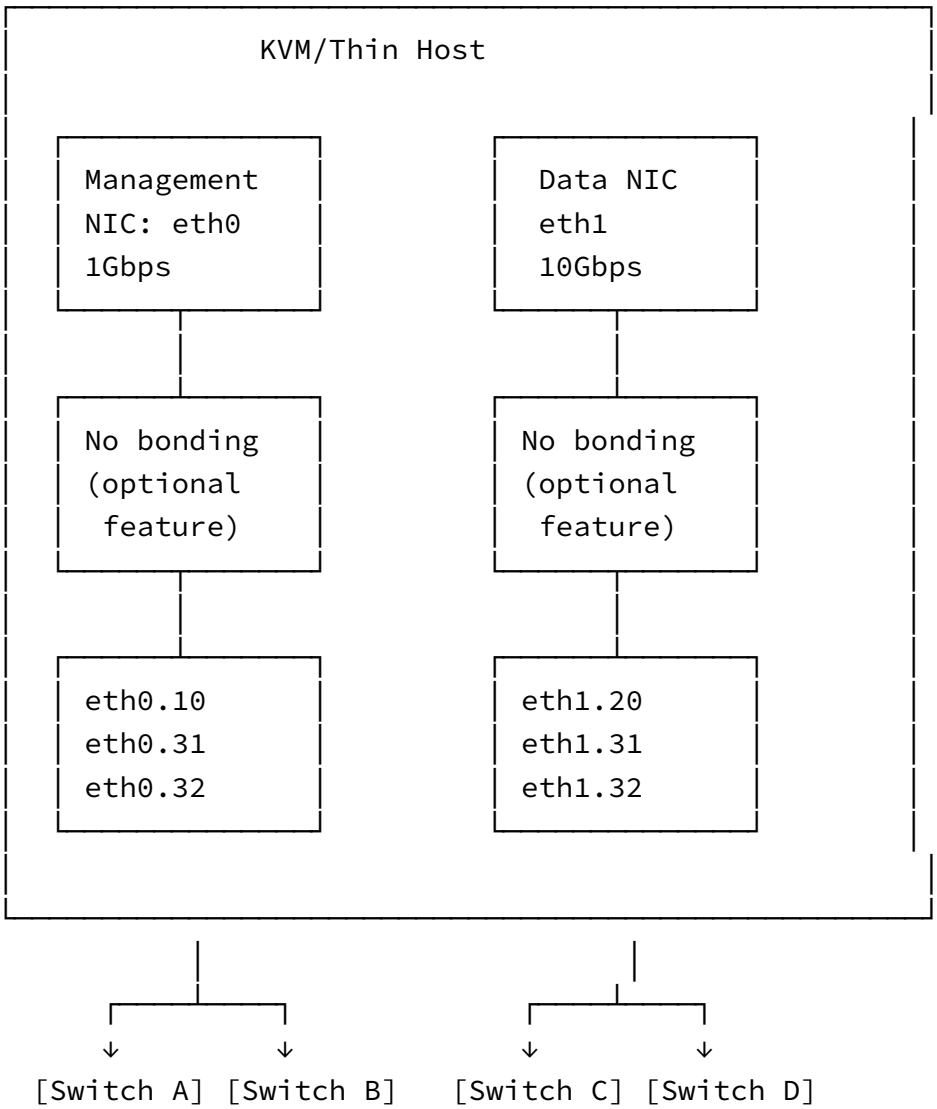
1. PXE boot from eth0 (untagged) → iPXE
2. iPXE creates eth0.10 (VLAN 10), gets IP
3. Downloads OS from VLAN 10
4. OS configures:
    - eth0.10 (management)
    - eth1.20 (VXLAN transport)
    - eth1.31, eth1.32 (storage)

10.4.3 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:

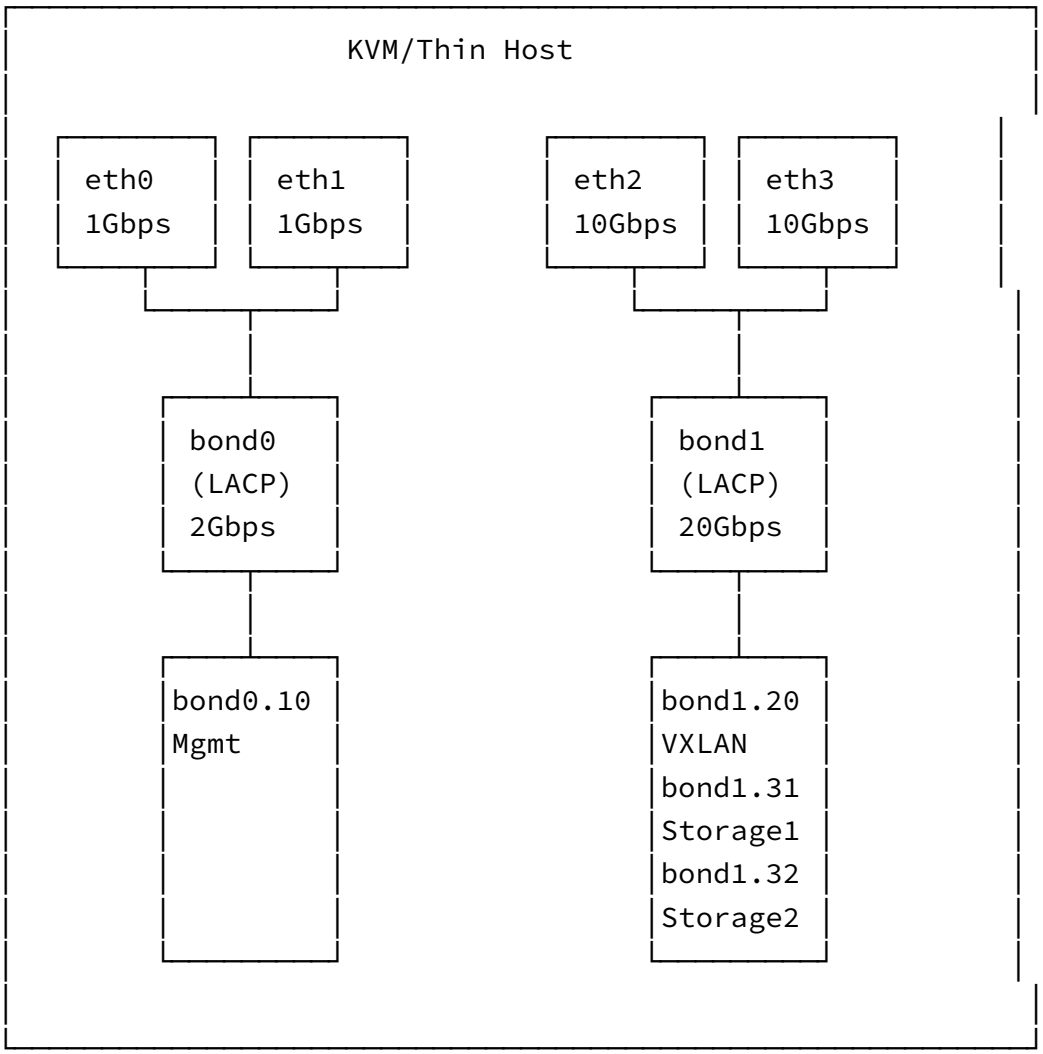
- Same as Profile 2
- Switches can be independent (no stacking/MC-LAG needed)
- If bonding used: same trunk configuration on both switches

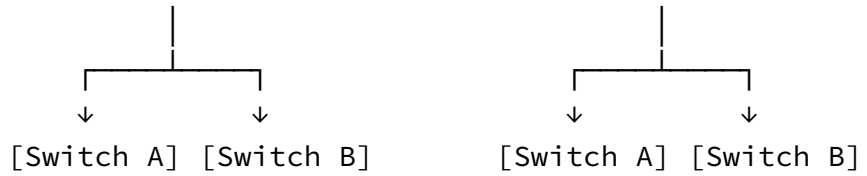
10.4.4 Profile 4: Quad NIC with LACP Bonding (Maximum Performance)

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.

10.4.5 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:

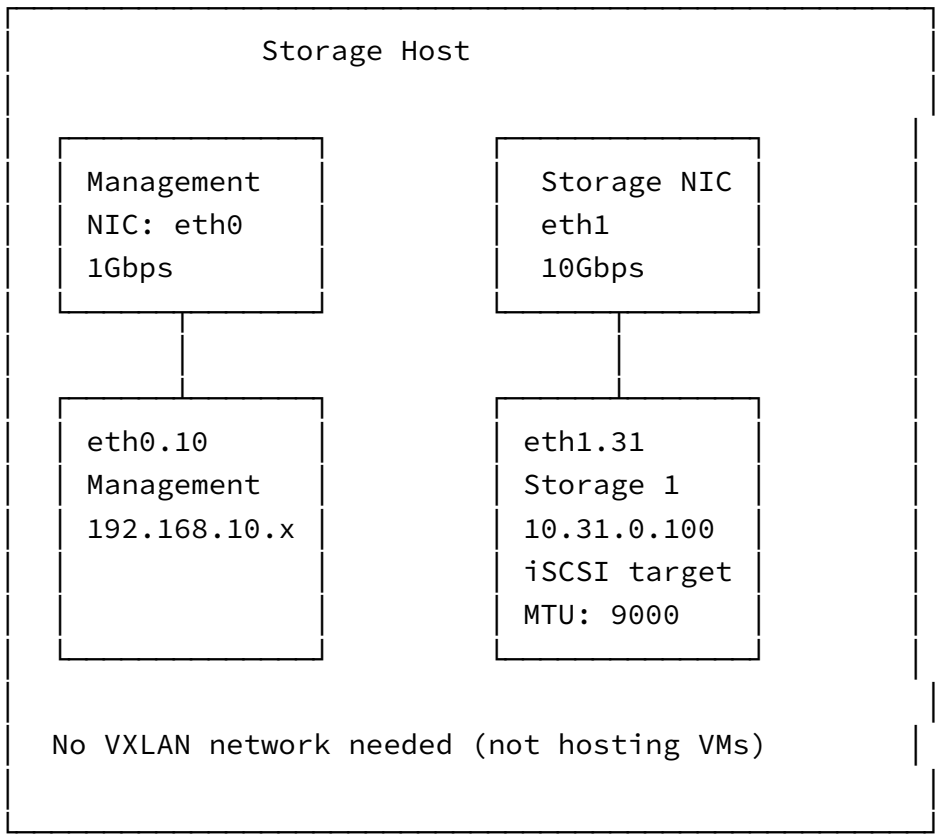
- More complex switch configuration
- Higher switch cost (enterprise feature)
- Vendor-specific setup

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.

10.5 Storage Host Considerations

10.5.1 Storage Host Network Configuration

Storage hosts (providing iSCSI targets) have similar but simplified networking:



VLANs used:

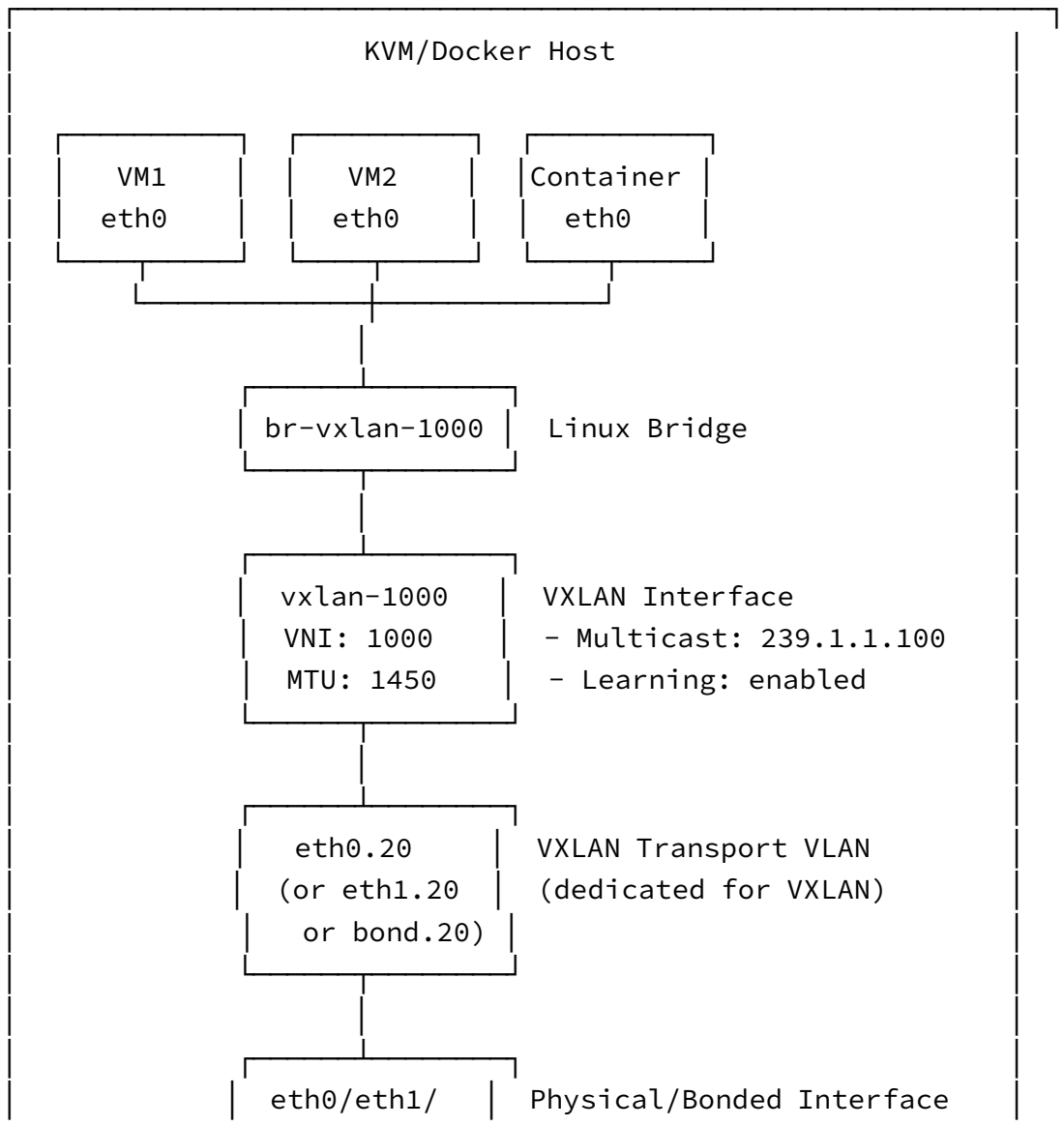
- VLAN 10: Management (SSH, monitoring)
- VLAN 31: Storage network for this host's iSCSI target
- No VLAN 20 (VXLAN) - storage hosts don't participate in customer networks

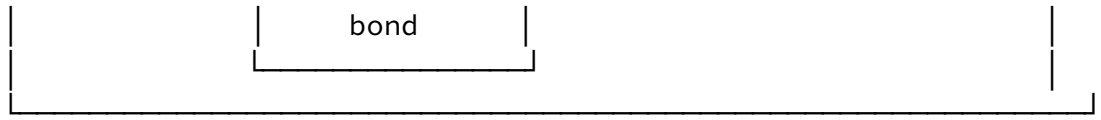
Each storage host uses one storage VLAN:

- 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.

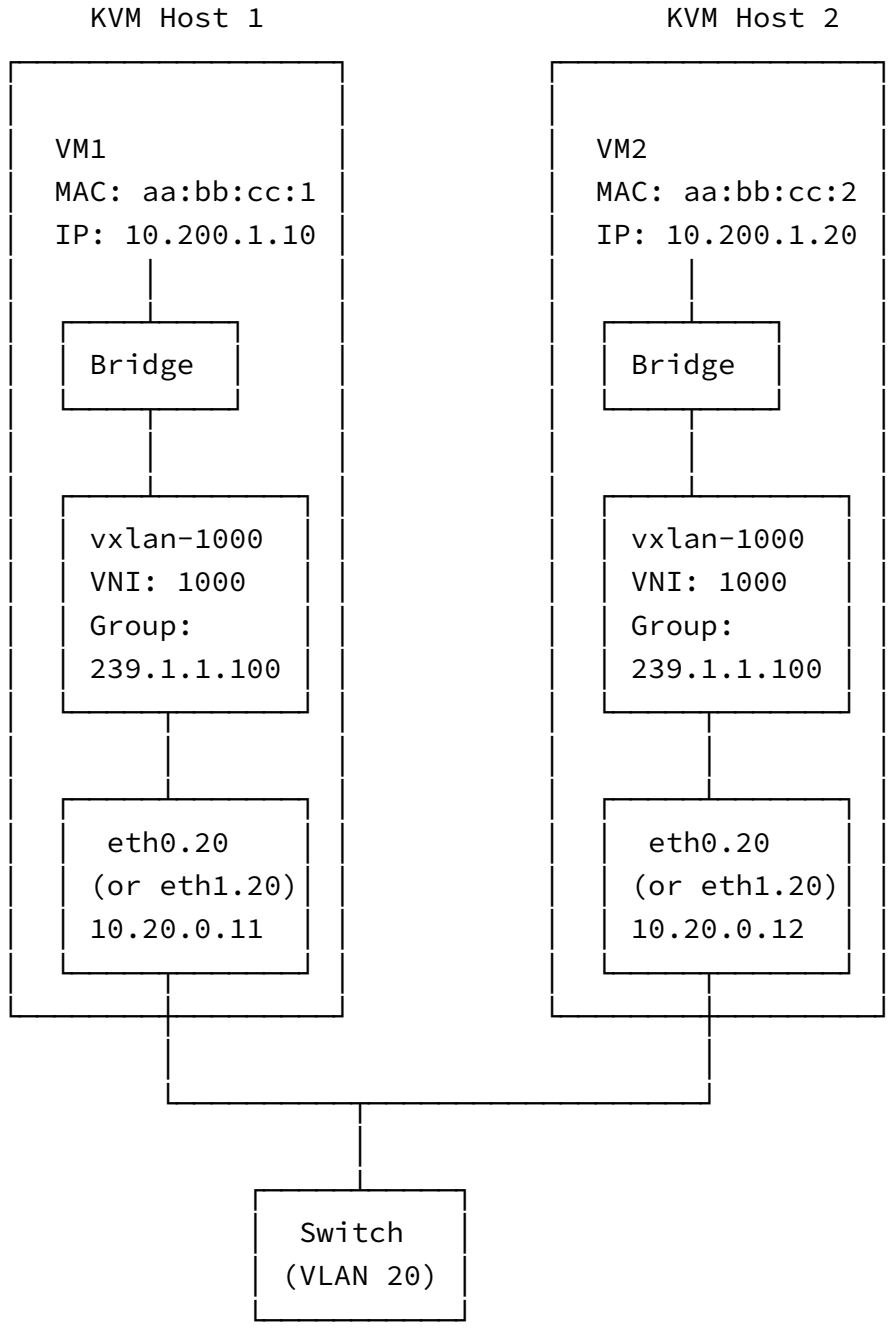
10.6 VXLAN Customer Network Architecture

10.6.1 Host-Level VXLAN Configuration





10.6.2 Multi-Host VXLAN Communication



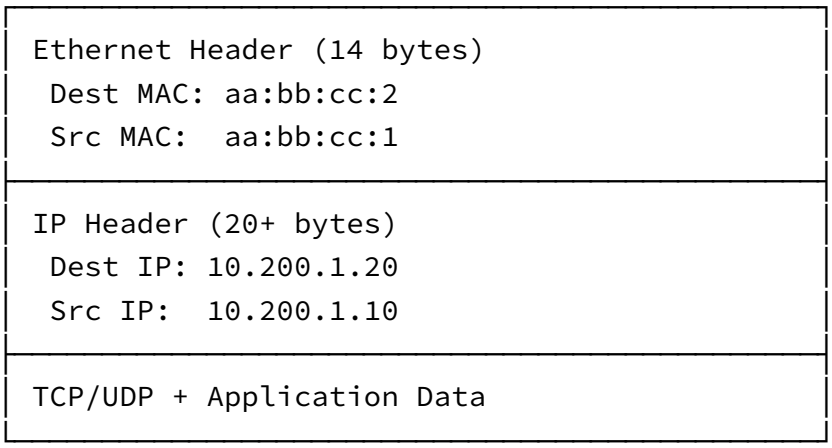
Communication Flow:

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

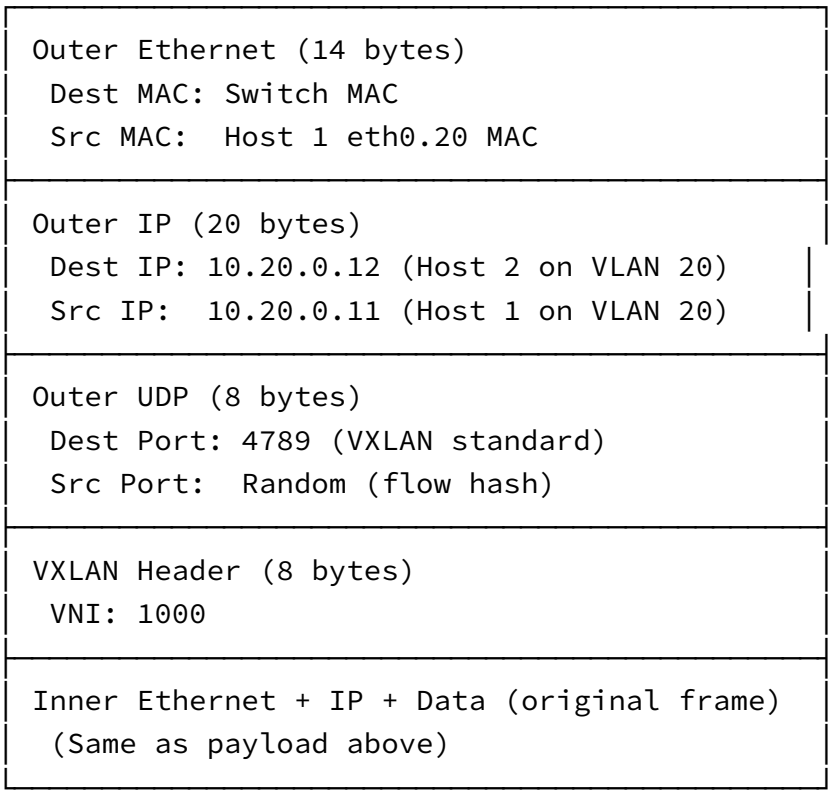
```
5. VXLAN overhead: 50 bytes (hence MTU 1450 for VMs)
```

10.6.3 VXLAN Packet Encapsulation

```
Inside VM (payload):
```




```
After VXLAN encapsulation (on wire):
```




```
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.

10.7 MTU Configuration Strategy

10.7.1 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
```

10.7.2 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

---

10.8 Switch Configuration Requirements

10.8.1 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)

10.8.2 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
```

10.8.3 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
```

10.8.4 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

10.9 Essential Configuration Commands

10.9.1 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
```

10.9.2 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
```

10.9.3 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
```

10.10 Network Component Summary

The following components require configuration management functions. Each represents a distinct functional area for automation:

10.10.1 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

10.10.2 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

10.10.3 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

10.10.4 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)

10.10.5 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

10.10.6 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

10.10.7 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

10.10.8 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

10.10.9 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)

10.10.10 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

10.10.11 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

10.10.12 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

10.10.13 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

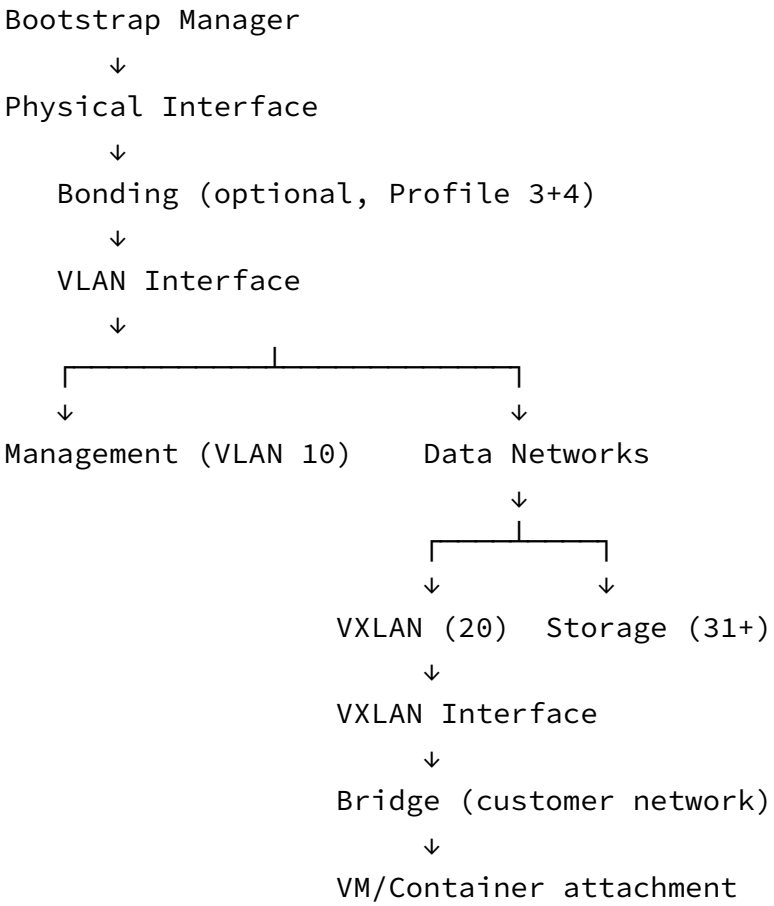
- Configure MTU on VM/container interfaces (1450 for VXLAN networks)
- Handle hot-plug/unplug if needed

Key considerations:

- Libvirt XML generation
- Docker network integration
- Customer network isolation enforcement
- MAC address management
- MTU settings (1450 for customer VMs on VXLAN)

10.11 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.

10.12 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

10.12.1 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.

11 Governance

11.1 AI Use Statement for HPS System Development

11.1.1 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.

11.1.2 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.

11.1.3 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.

11.1.4 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

11.1.5 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

11.1.6 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

11.1.7 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.

11.2 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.

11.2.1 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.

11.2.2 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.

12 HPS system documentation plan

12.1 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.

12.2 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.

12.3 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.

12.4 Functions reference

- Automatically generated from `lib/functions.d/` and other libraries.
- One function per page with purpose, arguments, usage examples, and related functions.

12.5 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.

12.6 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.

12.7 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.

12.8 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.

12.9 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.

12.10 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.

12.10.1 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.

12.10.2 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.

12.10.3 Docker

Link: https://docs.docker.com

Summary: Containerisation platform used to run the HPS environment (`hps-container`) in a controlled, reproducible manner.

12.10.4 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.

12.10.5 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.

12.10.6 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.

12.10.7 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.

12.10.8 Pandoc

Link: https://pandoc.org

Summary: Document converter used to compile HPS Markdown documentation into PDF, HTML, and other formats.

12.10.9 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.

12.10.10 Rocky Linux

Link: https://rockylinux.org

Summary: Enterprise-grade Linux distribution used as a primary OS target in HPS deployments.

12.10.11 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.

12.10.12 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.

12.10.13 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.