AUTOMATION FOR NETWORK INFRASTRUCTURE

IMPROVING AGILITY, SPEED, & PROCESSES
WITH OPEN SOURCE SOLUTIONS

Michael Ford
Senior Solutions Architect, Ansible
MANAGING NETWORKS HASN’T CHANGED IN 30 YEARS.
According to Gartner...

Automation Considerations

- Compute is no longer the slowest link in the chain
- Businesses demand that networks deliver at the speed of cloud
- Automation of repeatable tasks
- Bridge silos
Automation: SME as Code

- Leverages Human Experience
- Reduce Repetition
- Reduce Variability
- Reduce Isolation
Automation: SME as Code

- Leverages Human Experience
- Reduce Repetition
- Reduce Variability
- Reduce Isolation
Convert Procedures to Playbooks

1. Create VLAN
2. Add port to VLAN
3. Address Interface

Method of Procedure

- Define Intent, Policy, Architecture
- Apply across device type, vendor
Manage Lifecycle with Process & Playbooks

- Revision control, configuration management
- Ensure an ongoing steady-state
- Automated testing, reduce human error
Communicate with Playbooks
What is Ansible?

**Ansible** is a simple automation language that can perfectly describe an IT application infrastructure in Ansible Playbooks.

As a vendor agnostic framework Ansible can automate Arista (EOS), Cisco (IOS, IOS XR, NX-OS), Juniper (JunOS), Open vSwitch and VyOS.

**Ansible Engine** is an automation engine that runs Ansible Playbooks.

**Ansible Tower** is an enterprise framework for controlling, securing and managing your Ansible automation with a UI and RESTful API.
Why Ansible?

**SIMPLE**
- Human readable automation
- No special coding skills needed
- Tasks executed in order
- Get productive quickly

**POWERFUL**
- Image updates
- Configuration management
- Compliance
- Orchestrate the network lifecycle

**AGENTLESS**
- Agentless architecture
- Uses OpenSSH & WinRM
- No agents to exploit or update
- More efficient & more secure
The Flexibility of Choice

Business Requirements

Abstraction Through Automation

- BGP
- LB
- OSPF
- VLAN
- ACL
- QOS
- EVPN
- AAA
The Road To Automation

**STANDARDIZE**
with Red Hat Ansible Engine

- Snapshot State
- Detect Unauthorized Change
- Standardize Existing Configs
- Standardize New Deployments

**AUTOMATE**
with Red Hat Ansible Engine

- Automate common tasks
- Make changes across any set of network devices
- Validate that changes were successful

**SCALE**
with Red Hat Ansible Tower

- Automated deployment from Services Catalogue
- Automated compliance checking & enforcement
- API-Driven Integration with Application Development

Organize the Chaos ➔ Optimize your Infrastructure ➔ Stop Logging Into Devices
Improved Outcomes with Automation

<table>
<thead>
<tr>
<th>Time to Value</th>
<th>Time to Remediation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration &amp; Change Automation</td>
<td>Automated Fault Remediation</td>
</tr>
<tr>
<td>Faster Customer Service On-boarding</td>
<td>Faster Execution of Maintenance</td>
</tr>
<tr>
<td>Faster Execution of Change Requests</td>
<td>Faster Troubleshooting and Remediation</td>
</tr>
</tbody>
</table>
Playbooks & Network Modules
Under the Hood
Connection Plugins

Python code is executed locally on the control node

Python code is copied to the managed node, executed, then removed
Anatomy of a Playbook

```
- hosts: network

vars:
  site_domain_name: 'example.net'
  network_name_servers:
    - 8.8.8.8
    - 8.8.4.4
  log_host: 10.2.2.3

tasks:
  - name: Configure the hostname and domain name
    net_system:
      hostname: "{{ inventory_hostname }}"
      domain_name: "{{ site_domain_name }}"
      name_servers: "{{ network_name_servers }}"
  - name: configure host logging
    net_logging:
      dest: host
      name: "{{ log_host }}"
```

- **Inventory**: The devices to configure
- **Variables**: The key/value pairs that change from device to device
- **Tasks**: The tasks to perform on those devices
Network Functional Modules

Building Blocks

command
(e.g. ios_command)
- Executes command on device
- Provides output for further processing

config
(e.g. ios_config)
- Manipulates the config of the device
- Idempotent

facts
(e.g. ios_facts)
- Collects facts from the device
API-Driven Infrastructure

Well Defined, Role Based API

Easily Customizable Back End

Servers

Networking

Storage
Automate the Enterprise, not just Humans

- Test Network
- SMEs
- Operators

Dev | Ops

- Customers
- SoT
- Production Network

- Request
- Use
- Configure
- API
- UI

- Automate the Enterprise, not just Humans
- hosts: network
gather_facts: no
collection: local
tasks:
  - name: show version
    ios_command:
      commands:
        - show version
    wait_for:
      - result[0] contains Version
    register: results

- set_fact:
  ver: "{{ results.stdout[0]|regex_search('Version ([0-9.]+)', '\1') }}"

- debug: var=ver
Network Functional Module: Command

PLAY [network]
****************************************************************************************************
TASK [show version and show interfaces]
****************************************************************************************************
ok: [rtr1]

TASK [set_fact]
****************************************************************************************************
ok: [rtr1]

TASK [debug]  
ok: [rtr1] => {
    "ver": [
        "16.06.01"
    ]
}

PLAY RECAP  
rtr1: ok=3  changed=0  unreachable=0  failed=0
Network Functional Module: Config

- hosts: network
gather_facts: no
collection: local
tasks:
  - name: configure hostname
    ios_config:
      lines:
        - "hostname {{ inventory_hostname }}"
Network Functional Module: Config

First Run:
PLAY [network]
**********************************************************************************************
TASK [configure hostname]
**********************************************************************************************
changed: [rtr1]

PLAY RECAP
**********************************************************************************************
rtr1 : ok=1 changed=1 unreachable=0 failed=0

Second Run:
PLAY [network]
**********************************************************************************************
TASK [configure hostname]
**********************************************************************************************
ok: [rtr1]

PLAY RECAP
**********************************************************************************************
rtr1 : ok=1 changed=0 unreachable=0 failed=0
Network Functional Module: Facts

- hosts: network
  connection: local
  gather_facts: False
  tasks:

    - name: Get facts
      ios_facts:
        gather_subset: all

    - debug: msg="Serial Number is {{ ansible_net_serialnum }}"
Network Functional Module: Facts

PLAY [network]

TASK [Get facts]

ok: [rtr1]

TASK [debug]

ok: [rtr1] => {
  "msg": "Serial Number is 9G20X4MKLVP"
}

PLAY RECAP

rtr1 :
  ok=2  changed=0  unreachable=0  failed=0
Network Resource Modules

- name: configure the “management” vrf
  eos_vrf:
    name: management
    state: present
    when: ansible_network_os == 'eos'

- name: configure the “management” vrf
  ios_vrf:
    name: management
    description: oob mgmt vrf
    state: present
    when: ansible_network_os == 'ios'

- name: configure the “management” vrf
  nxos_vrf:
    name: management
    description: oob mgmt vrf
    state: present
    when: ansible_network_os == 'nxos'

- Per Platform Implementation
- Focused on managing a resource
- Declarative by design
- Handles complexity
Network Resource Modules

- name: configure network interface
  net_interface
    name: "{{ interface_name }}"
    description: "{{ interface_description }}"
    enabled: yes
    mtu: 9000
    state: up

- name: configure VLAN ID and name
  net_vlan:
    vlan_id: 20
    name: test-vlan
### Declarative Intent

- **name**: configure interface
- **net_interface**:
  - **name**: GigabitEthernet0/2
  - **description**: public interface configuration
  - **enabled**: yes
  - **state**: connected
  - **neighbors**:
    - **host**: core-01
      - **port**: Ethernet5/2/6
Aggregate Resources

- **name**: Configure VLANs
  
  ```yaml
ett_vlan:
    vlan_id: "{{ item.vlan_id }}"
    name: "{{ item.name }}"
    state: "{{ item.state | default('active') }}"
  with_items:
    - { vlan_id: 1, name: default }
    - { vlan_id: 2, name: Vl2 }
    - { vlan_id: 3, state: suspend }
```

Loop entries

- **name**: Configure VLANs and Purge
  
  ```yaml
ett_vlan:
    aggregate:
      - { vlan_id: 1, name: default }
      - { vlan_id: 2, name: Vl2 }
      - { vlan_id: 3, state: suspend }
    state: active
    purge: yes
```

Bulk entries

Multiple Operations

Single Operation
Applications Roles

- Focused on addressing operational use cases
- Approved and opinionated methods
- Developed, tested, and distributed by Ansible
- Agile development with gated release process
Software Supply Chain

Network Operators aren’t programmers, need one-stop for “approved” content

<table>
<thead>
<tr>
<th>Community</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where to obtain playbooks, roles, modules?</td>
<td>Trusted Distribution:</td>
</tr>
<tr>
<td>Who wrote them?</td>
<td>● Development: GitHub/ansible-network</td>
</tr>
<tr>
<td>Are they tested?</td>
<td>● Released: Ansible Galaxy</td>
</tr>
<tr>
<td>Who supports them?</td>
<td>Distributed CI test system</td>
</tr>
<tr>
<td></td>
<td>Supported by Red Hat</td>
</tr>
</tbody>
</table>
Automation for Teams

Ansible Tower technical introduction and overview
Ansible Tower is an **enterprise framework** for controlling, securing and managing your Ansible automation – with a **UI and RESTful API**.

- **RESTful API**
- **Role Based access control**
- **Deploy** entire applications with **push-button deployment** access
- All automations are **centrally logged**
RED HAT ANSIBLE TOWER
Scale + operationalize your automation

CONTROL

KNOWLEDGE

DELEGATION

RED HAT ANSIBLE ENGINE
Support for your Ansible automation

SIMPLE

POWERFUL

AGENTLESS

FUELED BY AN INNOVATIVE OPEN SOURCE COMMUNITY
Core Concepts & Best Practices
Layered Implementation
Simplifies playbooks, limits blast radius, and facilitates RBAC
Manage Applications, not Devices

- BGP
- OSPF
- NTP
- ACL
- AAA
- TRUNK
Inventory

[access_switches]
switch1
switch2

[access:vars]
ansible_network_os=ios

[routers]
juniper1 ansible_network_os=junos
cisco1 ansible_network_os=ios

[network:children]
access_switches
routers

- The devices being automated
- Part of SoT (Source of Truth).
- Static for ad-hoc activities and small environments.
- Dynamic for wider activities and large/enterprise/multi-site environments.
- Groups hosts by function, location, vendor, etc.
Directory Structure

Project Repository

- ansible.cfg
- inventory/
  - test/
    - hosts
    - host_vars/
    - group_vars/
  - prod/
    - hosts
    - host_vars/
    - group_vars/
- roles/
  - access_switch/
  - dist_router/
  - tenant_firewall/
- playbook1.yml
- playbook2.yml

Per-Environment Inventory and Data

Community/Organizational shared code

Repository-Specific Playbooks
Key/Value Pairs
Abstraction Through Data Models

**Cisco IOS**

```
router bgp 65082
no synchronization
bgp log-neighbor-changes
neighbor 10.11.12.2 remote-as 65086
no auto-summary
```

**Juniper JunOS**

```
bgp {
  local-as 65082;
  group TST {
    peer-as 65086;
    neighbor 10.11.12.2;
  }
}
```
Key/Value Pairs
Abstraction Through Data Models

**Cisco IOS**
```
router bgp 65082
no synchronization
bgp log-neighbor-changes
neighbor 10.11.12.2 remote-as 65086
no auto-summary
```
Key/Value Pairs
Abstraction Through Data Models

```yaml
bgp:
  global:
    config:
      as: 65082
  neighbors:
    neighbor:
      - neighbor_address: 10.11.12.2
        config:
          peer_group: TST
          peer_as: 65086
```

YANG OC Data Model

Vendor-Specific Rendering
The Flexibility of Ansible + Data Models

Any Model, Any Encoding, Any Transport

- Model
  - Vendor
  - OpenConfig
  - Custom

- Encoding
  - CLI
  - XML
  - JSON

- Transport
  - SSH
  - Netconf
  - API
Source of Truth (a.k.a. Key/Value Pairs)

Implementation

system:
  hostname: "{{ inventory_hostname }}"
  domain_name: eng.ansible.com

source_interface:
  name: Management1
  vrf: default

domain_lookup: no

name_servers:
  - 1.1.1.1
  - 2.2.2.2

vlan_data:
  - { id: 600, name: management }
  - { id: 601, name: users }

Operations

Engineering

Definition

Production

Infrastructure

Applications

Servers

Storage

Network

Desired State
Facts: Loading and Using

Load SoT from Inventory:

```yaml
host_vars\switch1\interfaces.yml
```

or

![CMDB](image)

or

Manually load w/Playbook:

```yaml
- include_role:
  name: load_interface_data
```

Available for Playbooks to reference:

```yaml
- name: Set Interface Attributes
  net_interface
  name: "{{ item }}"
  description: "{{ item.description }}"
  enabled: "{{ item.enabled }}"
  with_items: "{{ interfaces.keys() }}"
```

Per-Inventory Item Facts Cache
hostvars[inventory_hostname]:
  interfaces:
    Gi1/0/1:
      description: "ht3-node1:eth0"
      enabled: True
      mtu: 1500
      mode: trunk
      native_vlan: 99
    Gi1/0/2:
      description: "ht3-node2:eth0"
      enabled: True
      mtu: 1500
      mode: access
      access_vlan: 10
    Gi1/0/3:
      description: "ht3-node3:eth0"
      enabled: True
      mtu: 1500
      mode: access
      access_vlan: 10

Per-Inventory Item Facts Cache

Playbook writes out to inventory:

- name: write out the interfaces vars
  copy:
    dest: "{{ inventory_dir }}/{{ inventory_hostname }}/interfaces.yml"
    content: "{{ interfaces | to_nice_yaml }}"

or write out to CMDB

- include_role:
  name: save_to_cmdb
Roles

Roles are ways of automatically loading certain vars_files, tasks, and handlers based on a known file structure. Grouping content by roles also allows easy sharing of roles with other users.

```yaml
set of tasks to achieve a function

ios_command
  ...
ios_vlan
  ...
ios_interface

include_role:
  name: access_switch

Re-usable, Testable function available to others
```
Testing Roles

- hosts:
  access_switches
roles:
  - access_switch

Switch by specifying inventory

[access_swicthes]

Test

Prod

[access_swicthes]
The Automated Enterprise

SMEs

Operators

Developers

Develop

Plan

Test

Deploy

Operate

SoT

ANSIBLE TOWER by Red Hat

52 INSERT DESIGNATOR, IF NEEDED
Automation for Teams

Ansible Tower technical introduction and overview
RED HAT ANSIBLE TOWER
Scale + operationalize your automation

CONTROL  KNOWLEDGE  DELEGATION

RED HAT ANSIBLE ENGINE
Support for your Ansible automation

SIMPLE  POWERFUL  AGENTLESS

FUELED BY AN INNOVATIVE OPEN SOURCE COMMUNITY
 Ansible Playbooks

Ansible Tower
- Role-based Access Control
- Knowledge & Visibility
- Scheduled & Centralized Jobs
  - Simple User Interface
  - Tower API

Ansible Engine
- Open Source Module Library
- Python Codebase
- Plugins

Automate Your Enterprise
- Infrastructure
  - Linux, Windows, Unix...
- Networks
  - Arista, Cisco, Juniper...
- Containers
  - Docker, LXC...
- Cloud
  - AWS, Google Cloud, Azure...
- Services
  - Databases, Logging, Source Control Management...

Use Cases
- Provisioning
- Configuration Management
- App Deployment
- Continuous Delivery
- Security & Compliance
- Orchestration

Transport
- SSH, WinRM, etc.
Heads-up NOC-style **automation dashboard** displays everything going on in your Ansible environment.
Anssible Tower

Activity Stream

Securely stores every Job that runs, and enables you to view them later, or export details through Tower's API.
Ansible Tower

Multi-Playbook Workflows

Tower’s multi-Playbook workflows chains any number of Playbooks together to create a single workflow. Different Jobs can be run depending on success or failure of the prior Playbook.
Ansible Tower
Scale-Out Clustering

Connect multiple Tower nodes into a Tower cluster to add redundancy and capacity to your automation platform.

Add reserved capacity and capacity by organization, and deploy remote execution nodes for additional local capacity.
Ansible Tower
Manage and Track Your Inventory

Tower’s **inventory syncing** and **provisioning callbacks** allow nodes to request configuration on demand, enabling autoscaling.

Smart Inventories allow you to organize and automate hosts across all your providers based on a powerful host fact query engine.

See alerts from Red Hat Insights directly from Tower, and use Insights-provided Playbook Remediation to fix issues in your infrastructure.
Ansible Tower
Schedule Jobs

Enables you to schedule any Job now, later, or forever.
Ansible Tower
Integrated Notifications

Stay informed of your automation status via integrated notifications. Connect Slack, Hipchat, SMS, email and more.
Ansible Tower

Self-Service IT

Tower lets you launch Playbooks with just a single click. It can prompt you for variables, let you choose from available secure credentials and monitor the resulting deployments.
Ansible Tower

External Logging

Connect Tower to your external logging and analytics provider to perform analysis of automation and event correlation across your entire environment.
Ansible Tower

Network Visualization

**DISCOVER**
Know what network devices and services are installed, represented visually

**DESIGN**
Create and build new topologies, adapt existing topologies from discovery, and utilize existing playbooks

**DEPLOY**
Convert designs to actual physical or virtual deployments using Ansible playbooks and network modules, and then automate deployment

NOTE: Currently in Alpha and not committed to a release
Use Cases
Automating Complex Tasks

1. Automate the deployment of the individual components as a workflow.
2. Make that workflow available to operators.
3. Force changes to workflow to maintain compliance
4. Run that workflow on a regular bases to detect any deviation from the original deployment.
Automating Troubleshooting

collect:
  ios_router:
    - show ip ospf neighbors....
    - show bgp summary....
    - show ip ospf route....
    - show ip bgp route....
  nxos_switch:
    - show ip arp....
    - show mac address-table....
  bigip:
    - ....
  junos:
    - ....
  linux:
    - ....
DC Fabric Deployment

interfaces:
  vtep:
    name: nve1
    source_interface: loopback0
    host_reachability: yes

control:
  name: loopback0
  address: "{{ control_plane_address }}"

fabric:
  Ethernet1/1-4:
    name: Ethernet1/1-4
fw_rules:
- { rule: "public", src_ip: 0.0.0.0/0, dst_ip: 192.133.160.23/32, dst_port: 32400, proto: tcp, action: allow, comment: app1 }
- { rule: "public", src_ip: 0.0.0.0/0, dst_ip: 192.133.160.23/32, dst_port: 1900, proto: udp, action: allow, comment: app2 }
- { rule: "public", src_ip: 0.0.0.0/0, dst_ip: 192.133.160.23/32, dst_port: 3005, proto: tcp, action: allow, comment: app3 }
- { rule: "public", src_ip: 0.0.0.0/0, dst_ip: 192.133.160.23/32, dst_port: 5353, proto: udp, action: allow, comment: app4 }

- name: Insert ASA ACL
  asa_config:
    lines:
    - "access-list {{ item.rule }} extended {{ item.action | default (omit) }} {{ item.proto | default (omit) }} {{ item.src_ip | ipaddr('network') }} eq {{ item.dst_port }}"

- name: Create security rules
  panos_security_rule:
    operation: "{{ item.action | default (omit) }}"
    rule_name: "{{ item.comment | default (omit) }}"
    service: "{{ item.dst_port | default (omit) }}"
    description: "{{ item.description | default (omit) }}"
    source_zone: "{{ item.rule | default (omit) }}"
    destination_zone: "{{ item.destination_zone | default (omit) }}"
    action: "{{ item.action | default ('allow') }}"
    commit: "{{ item.comment | default (omit) }}"
Hybrid Cloud

1. Automate the creation of the VPC and network components.
2. Deploy the same routers, load-balancers, and firewalls that you use on-site.
3. Automate the entire network in a uniform way.
Workflow Automation

1. Customer makes request from the service catalog
2. Request goes through approval process
3. Service catalog calls Tower API to fulfill request
4. Ansible Tower updates ticket
Tier 1 Support Automation

1. Monitoring/Logging Platform detects event and calls the Ansible Tower API
2. Ansible Tower runs a playbook to collect event-specific information
3. Ansible Tower runs a playbook to open a support ticket and/or notify Tier 2 support
THANK YOU

plus.google.com/+RedHat
linkedin.com/company/red-hat
youtube.com/user/RedHatVideos
facebook.com/redhatinc
twitter.com/RedHatNews
How Ansible Works

**ANSIBLE’S AUTOMATION ENGINE**

- **PUBLIC / PRIVATE CLOUD**
- **CMDB**
- **PUBLIC / PRIVATE CLOUD**

**USERS**

**ANSIBLE PLAYBOOK**

**INVENTORY**

**API**

**MODULES**

**PLUGINS**

**HOSTS**

**NETWORK DEVICES**

**CLOUD**

OpenStack, VMware, EC2, Rackspace, GCE, Azure, Spacewalk, Hanlon, Cobbler

**CUSTOM CMDB**
AUTOMATION FOR TEAMS
Ansible Tower technical introduction and overview
WHAT IS ANSIBLE TOWER?

Ansible Tower is an enterprise framework for controlling, securing and managing your Ansible automation – with a UI and RESTful API.

• Role-based access control

• Deploy entire applications with push-button deployment access

• All automations are centrally logged
## AUTOMATION and SDN

### BENEFIT

<table>
<thead>
<tr>
<th>Benefit</th>
<th>SDN</th>
<th>AUTOMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reconfigure the network from a central point</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Reduced vendor lock in with commodity hardware</td>
<td>?</td>
<td>✔</td>
</tr>
<tr>
<td>Leverage existing infrastructure</td>
<td>☒</td>
<td>✔</td>
</tr>
<tr>
<td>Programmability</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Reduced opex/capex costs</td>
<td>?</td>
<td>✔</td>
</tr>
</tbody>
</table>
PRIMARY SDN USE CASES

- Data Center Tenancy, Microsegmentation
- Virtualization and Cloud Agility
- Automation and Programmability
- VLAN Sprawl and Complexity
- Application/Infrastructure Provisioning
- Network Scaling to support workloads
- Lower CapEx (hardware costs)
- Simplify Adds, Moves, Changes
SOFTWARE DEFINED NETWORK (SDN)

Ref: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf
RED HAT ANSIBLE ENGINE NETWORKING ADD-ON

NETWORK MODULES

• Developed, maintained, tested, and supported by Red Hat

• 140+ supported modules and growing*

• Red Hat reports and fixes problems

• Networking modules included with Ansible Engine offering, but the Ansible Engine Networking Add-On SKU purchase is required for full support

*take special note of the specific supported platforms

NETWORKING ADD-ON INCLUDED SUPPORT:
Arista EOS
Cisco IOS
Cisco IOS XR
Cisco NX-OS
Juniper Junos
Open vSwitch
VyOS
NETWORK VISUALIZATION (USE CASES)

DISCOVER
Know what network devices and services are installed, represented visually.

DESIGN
Create and build new topologies, adapt existing topologies from discovery, and utilize existing playbooks.

DEPLOY
Convert designs to actual physical or virtual deployments using Ansible playbooks and network modules, and then automate deployment.

NOTE: Currently in Alpha and not committed to a release.

Group, Copy/Paste, Zoom
Drag and Drop roles into a device
- device which roles are used where

Look at physical connections
- see this device’s perspective of the network
"REGIONAL" SITE VIEW

Geographically keep track of sites - real world mapping at a high level
Automating Complex Tasks: Networks

Problem:

- Deploying, configuring, and maintaining a network requires many manual tasks by skilled artisans. Configuration issues and unknown changes account for a majority of downtime.
Firewall/Load Balancer Updates

Problem:

- Rapid Application development requires many updated to firewalls and load balancers. Manually adding these takes time and is prone to error.
- The task is made more difficult when multiple vendors are deployment.
Hybrid Cloud

Problem:

- Public/Hybrid cloud increases the number of things to manage
- Cloud things are different than on-prem things and different between clouds increasing complexity
Workflow Automation

Problem:

• Most enterprises have a ticketing/ approval system for common IT tasks. Once the task goes through the approval process, it ends up in a person’s queue for manual action.
Problem:

• Many enterprises monitor for error conditions, but most don’t do anything with them. If they do, there is no good data to figure out the problem.