

From Scripts to Scale: Transforming Enterprise Automation with OPS/MVS and Ansible

Paul Buchanan, Principal Mainframe Systems Programmer
paul.buchanan@ensono.com

Jan Prihoda, Product Owner
jan.prihoda@broadcom.com

Cody Giardinello, Engineering Supervisor
cody.giardinello@broadcom.com

Ansible's Journey to z/OS

2012

- Founded by Michael DeHaan with origins in simplicity

2015

- **Red Hat acquired Ansible** accelerating enterprise adoption

2018

- **IBM acquired Red Hat** bringing Ansible into the IBM ecosystem

Today

- **Ansible extends to z/OS** through the IBM Z Ansible Core Collection

What is Ansible?

A Powerful Automation Engine

Simple & Human Readable

- Uses YAML playbooks—no complex scripts

Agentless

- No software to install on target systems—just SSH

Idempotent

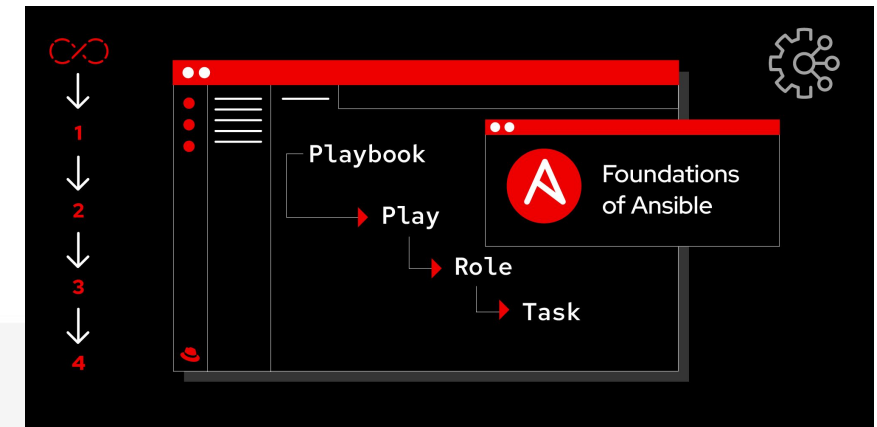
- Runs tasks only when needed, preventing unnecessary changes

Platform-agnostic

- Works across Linux, Windows, cloud, and z/OS

Extensible Ecosystem

- Supports plugins, modules, and integrations with tools



Using Ansible for Mainframe Tasks

A Simple, Structured Syntax



```
// Query jobs beginning with 'JOB12'
```

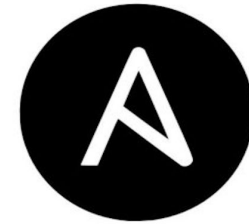
```
zowe jobs ls jobs -o "*" -p "JOB12*"
```

Using Ansible for Mainframe Tasks

A Simple, Structured Syntax



```
// Query jobs beginning with 'JOB12'  
zowe jobs ls jobs -o "*" -p "JOB12*"
```

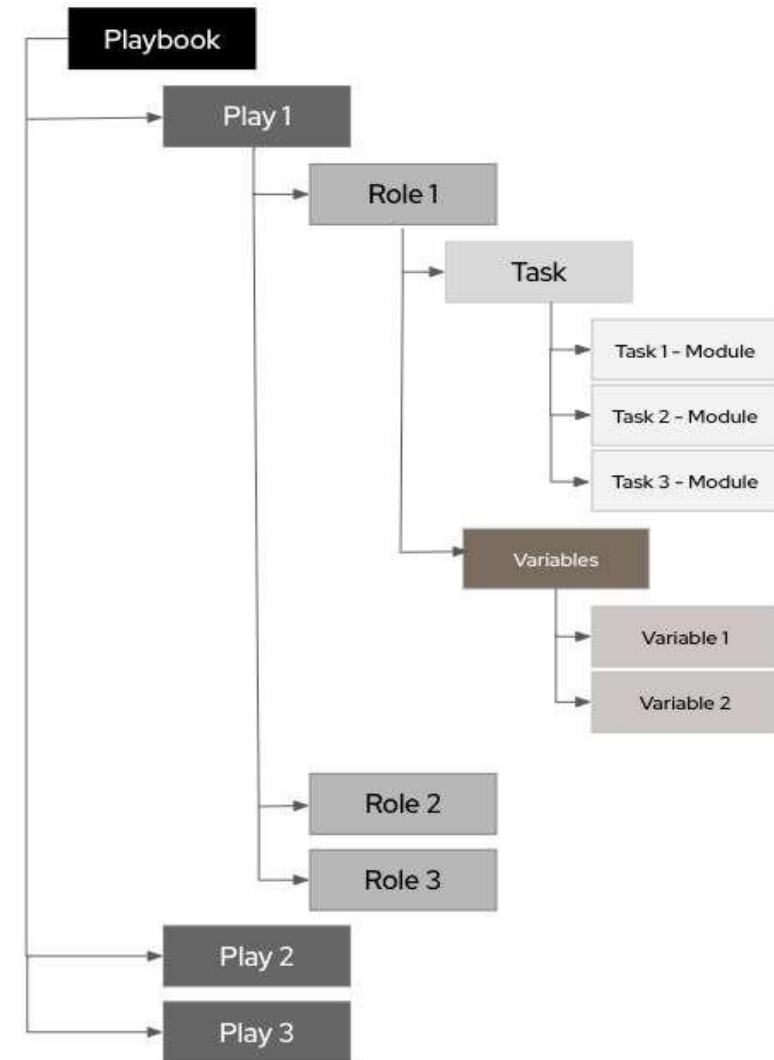
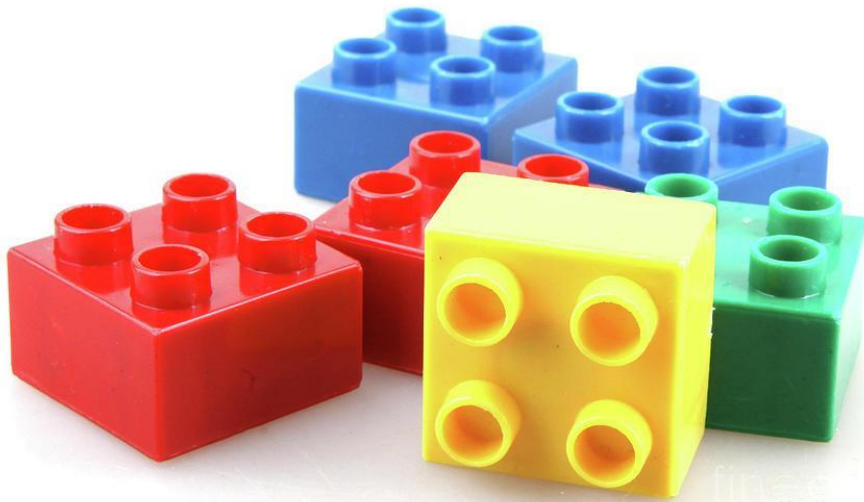


ANSIBLE

```
- name: Query jobs beginning with 'JOB12'  
zos_job_query:  
  job_id: "JOB12*"
```

Using Ansible for Mainframe Tasks

Robust, repeatable, and secure





Ansible at Ensono

| Ansible at Ensono

Ansible for Mainframe

How Ensono leverages Ansible automation to modernize, secure, and simplify mainframe operations — reducing manual effort and empowering teams at every skill level.

3+

Core Automation Use Cases

↓ **Risk**

Consistent, Repeatable Processes

Faster

Renewal & Upgrade Cycles

Simpler

Modern Tooling for All Skill Levels

Top Ansible Use Cases for Mainframe

Ensono's proven automation playbooks driving efficiency across mainframe environments

| Ansible at Ensono

Ensono's proven automation playbooks driving efficiency

**SSL Certificate Renewal**
SECURITY AUTOMATION

Expired SSL certificates on mainframes can cause critical outages and compliance violations. Ensono uses Ansible playbooks to automate the end-to-end SSL certificate renewal process, ensuring secure, uninterrupted operations.

- ✓ Automated discovery and tracking of certificate expiry dates
- ✓ Scheduled renewal workflows triggered well ahead of expiration
- ✓ Consistent certificate deployment across all mainframe environments
- ✓ Reduces risk of human error and missed renewal windows
- ✓ Audit-ready logs for every renewal action taken

| Ansible at Ensono

Ensono's proven automation playbooks driving efficiency



SSL Certificate Renewal

SECURITY AUTOMATION

Expired SSL certificates on mainframes can cause critical outages and compliance violations. Ensono uses Ansible playbooks to automate the end-to-end SSL certificate renewal process, ensuring secure, uninterrupted operations.

- ✓ Automated discovery and tracking of certificate expiry dates
- ✓ Scheduled renewal workflows triggered well ahead of expiration
- ✓ Consistent certificate deployment across all mainframe environments
- ✓ Reduces risk of human error and missed renewal windows
- ✓ Audit-ready logs for every renewal action taken



License Key Renewals

COMPLIANCE AUTOMATION

Managing software license keys across large mainframe estates is complex and error-prone. Ensono automates license key renewals with Ansible, keeping mainframe software compliant and fully operational without manual intervention.

- ✓ Centralized tracking of license expiration across all mainframe systems
- ✓ Automated key distribution and application via playbooks
- ✓ Proactive alerts before licenses lapse to avoid disruptions
- ✓ Eliminates emergency manual processes tied to last-minute renewals
- ✓ Supports compliance reporting and governance requirements

Ansible at Ensono

Ensono's proven automation playbooks driving efficiency



SSL Certificate Renewal

SECURITY AUTOMATION

Expired SSL certificates on mainframes can cause critical outages and compliance violations. Ensono uses Ansible playbooks to automate the end-to-end SSL certificate renewal process, ensuring secure, uninterrupted operations.

- ✓ Automated discovery and tracking of certificate expiry dates
- ✓ Scheduled renewal workflows triggered well ahead of expiration
- ✓ Consistent certificate deployment across all mainframe environments
- ✓ Reduces risk of human error and missed renewal windows
- ✓ Audit-ready logs for every renewal action taken



License Key Renewals

COMPLIANCE AUTOMATION

Managing software license keys across large mainframe estates is complex and error-prone. Ensono automates license key renewals with Ansible, keeping mainframe software compliant and fully operational without manual intervention.

- ✓ Centralized tracking of license expiration across all mainframe systems
- ✓ Automated key distribution and application via playbooks
- ✓ Proactive alerts before licenses lapse to avoid disruptions
- ✓ Eliminates emergency manual processes tied to last-minute renewals
- ✓ Supports compliance reporting and governance requirements



Python & ZOAU Upgrades

CORE ANSIBLE DEPENDENCIES

Python and ZOAU (Z Open Automation Utilities) are foundational dependencies that power IBM's Ansible collections for Z. Keeping them current is critical. Ensono automates these upgrades to maintain a healthy, modern Ansible runtime on the mainframe.

- ✓ Automated detection of outdated Python and ZOAU versions
- ✓ Controlled upgrade playbooks with pre- and post-validation checks
- ✓ Minimizes downtime with staged rollout across environments
- ✓ Ensures compatibility with the latest IBM Z Ansible collections
- ✓ Reduces barrier for teams adopting Ansible on the mainframe



Basic Setup

| Configure the Control Node

Install Prerequisites



| Configure the Control Node

Install Prerequisites



```
sudo apt install python3
```



| Configure the Control Node

Install Prerequisites



```
sudo apt install python3
```



```
sudo python -m pip install ansible
```



Configure the Control Node

Install Prerequisites



```
sudo apt install python3
```



```
sudo python -m pip install ansible
```



```
sudo apt install openssh-client
```

| Configure the Control Node

Install Prerequisites



```
sudo python -m pip install ansible
```

Configure the Control Node

Install Prerequisites



ANSIBLE

```
sudo python -m pip install ansible
```

IBM. `ibm.ibm_zos_core`

```
ansible-galaxy collection install ibm.ibm_zos_core
```

Configure the Control Node

Install Prerequisites



ANSIBLE

```
sudo python -m pip install ansible
```

 **ibm.ibm_zos_core**

```
ansible-galaxy collection install ibm.ibm_zos_core
```



broadcom.ops

```
ansible-galaxy collection install broadcom.ops
```

Ansible Collections

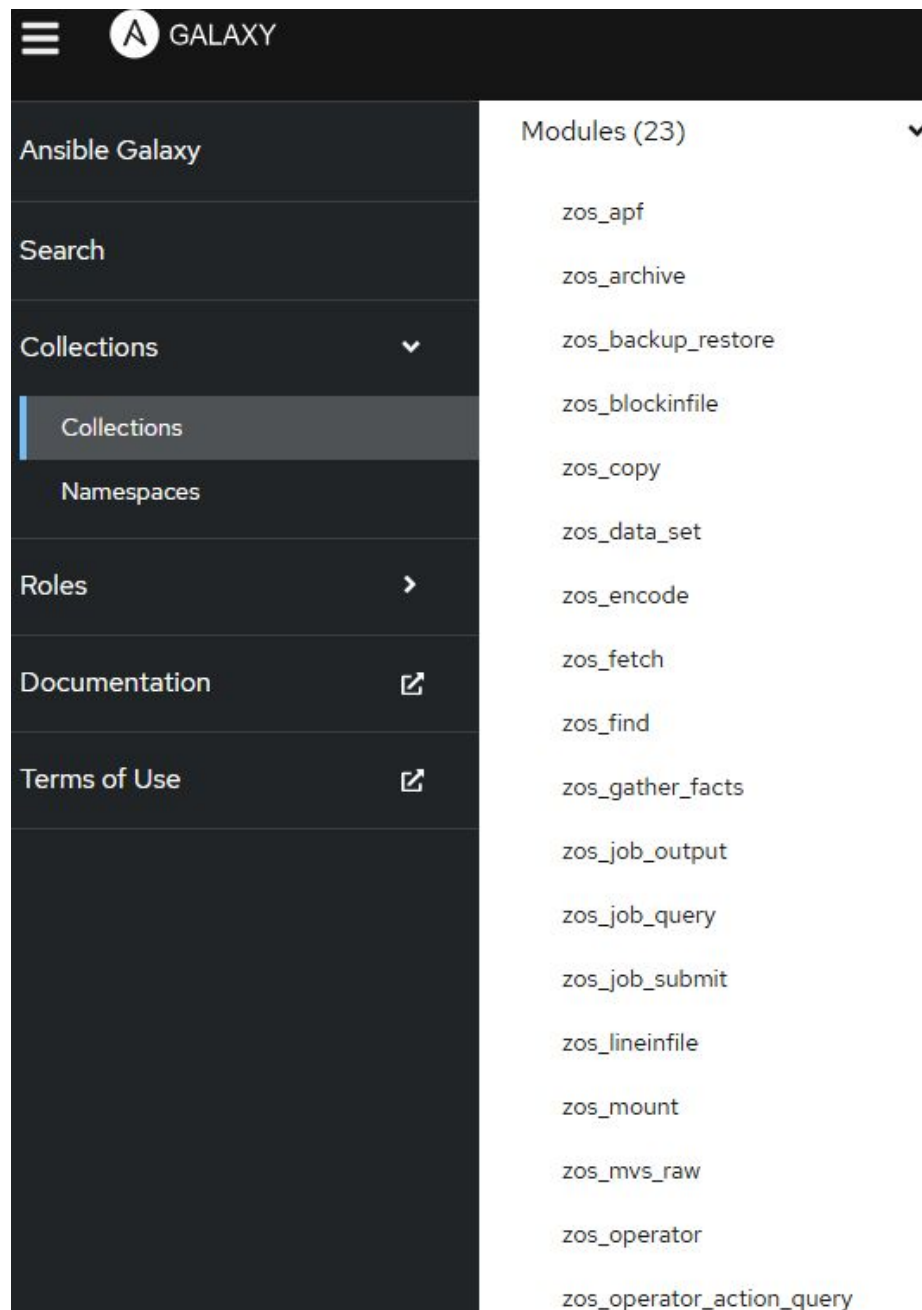
Open Source and Supported



 **ibm.ibm_zos_core**



broadcom.ops



Ansible Collections

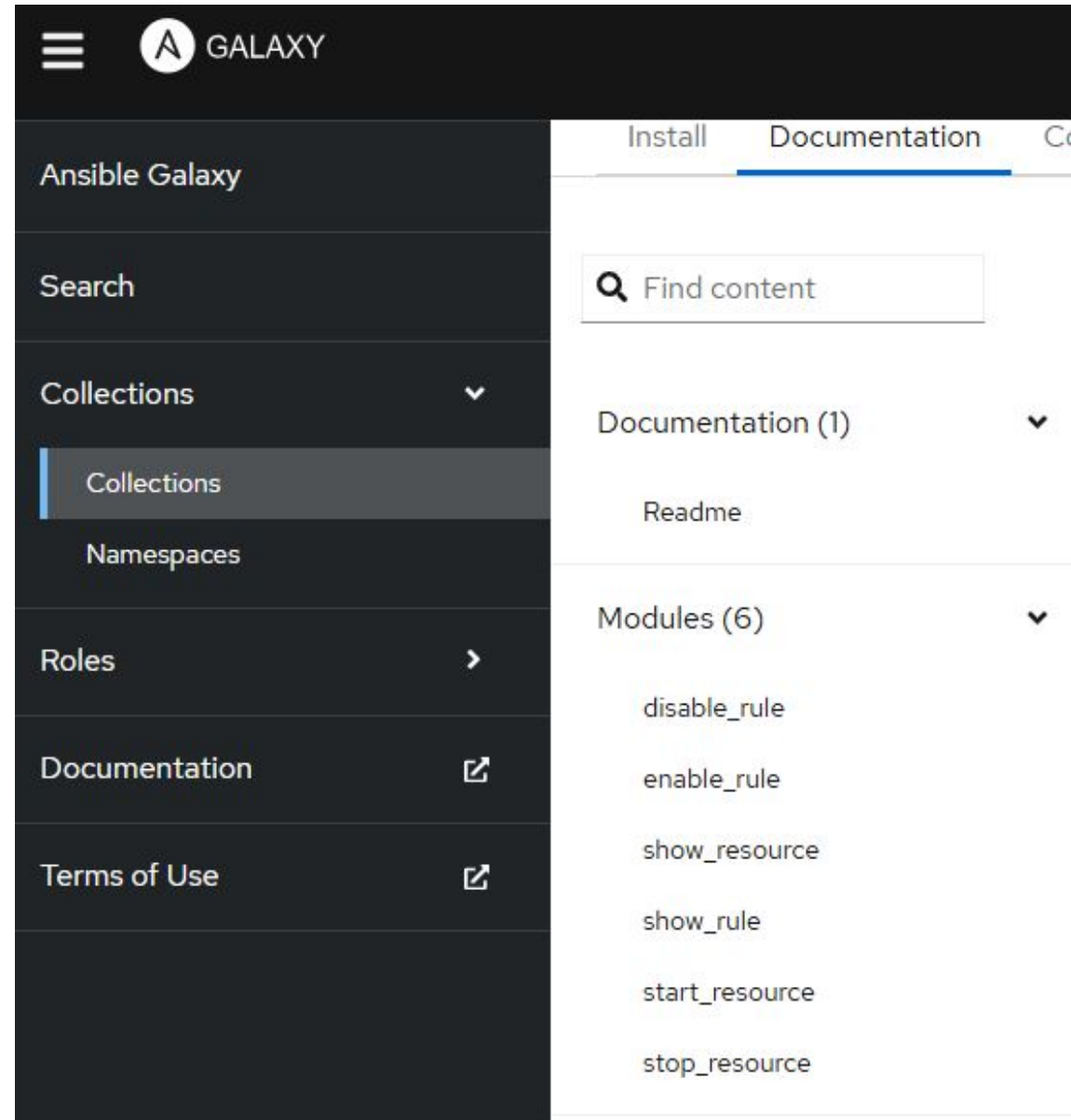
Open Source and Supported



IBM `ibm.ibm_zos_core`



`broadcom.ops`

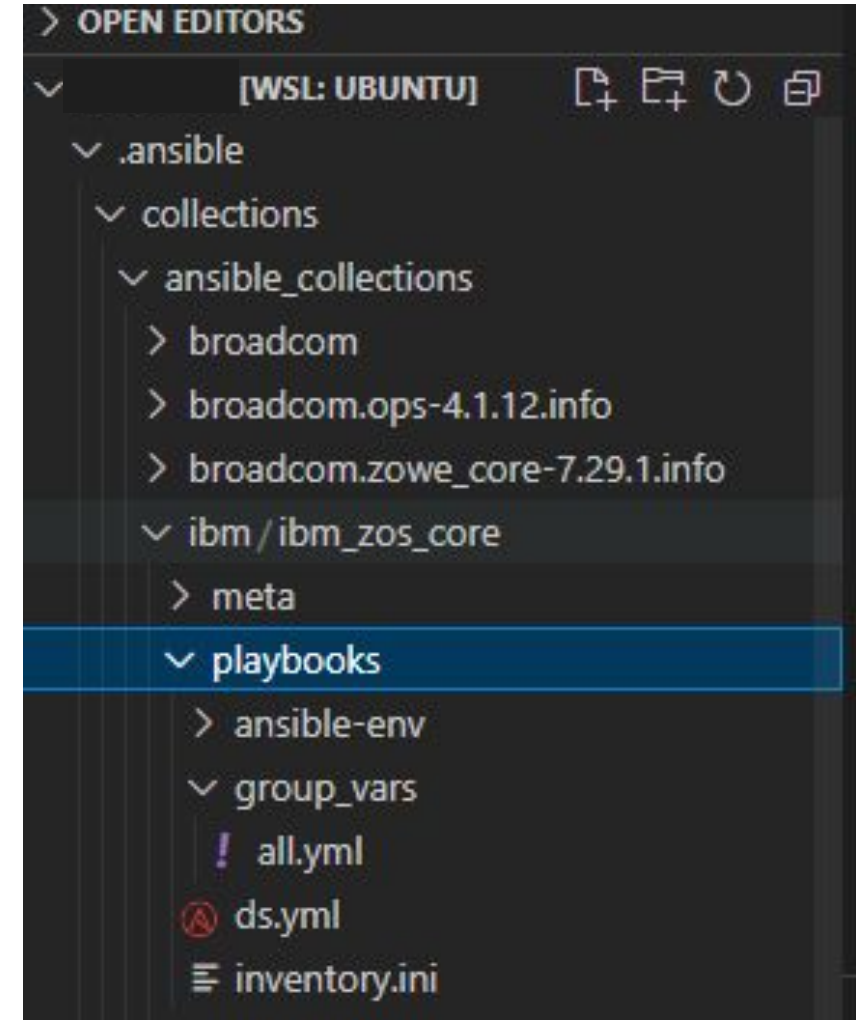




Our First Playbook

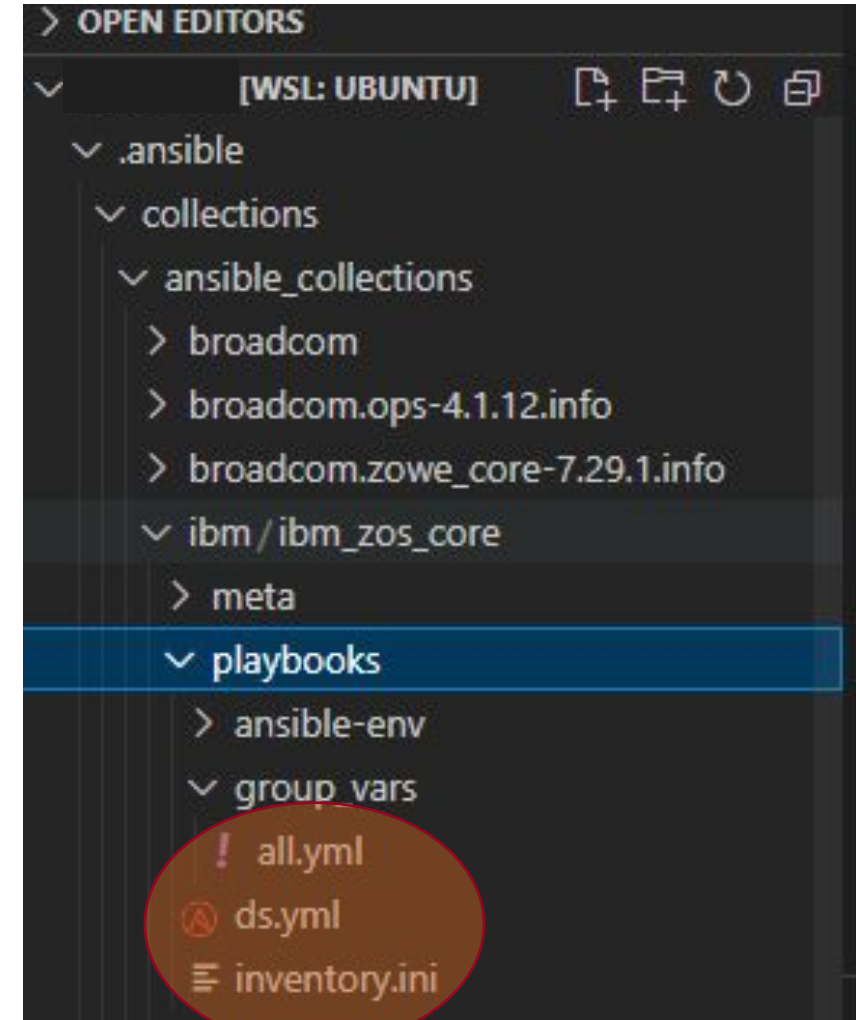
Our First Playbook

A Three Pronged Approach



Our First Playbook

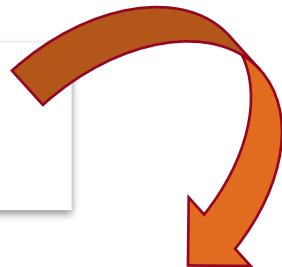
A Three Pronged Approach



| Inventory File

Connecting to z/OS Managed Nodes

inventory.ini

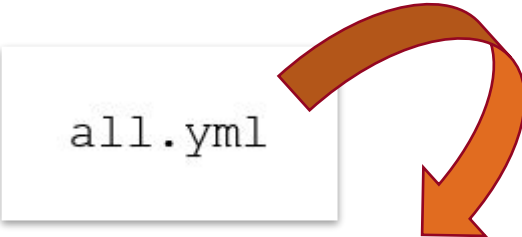


```
.ansible > collections > ansible_collections > ibm > ibm_zos_core > playbooks > ≡ inventory.ini
1  [zsystem]
2  ansible_host=      ansible_user=      ansible_python_interpreter=/usr/lpp/IBM/cyp/v3r12/pyz/bin/python3
```

Environment Variable File

Shared Among Playbooks

all.yml



Ansible Vars File - Ansible variables File (vars.json)

environment_vars:

_BPXK_AUTOCVT: ON

ZOAU_ROOT: '/usr/lpp/IBM/zoau/v1r3'

PYTHONPATH: '/usr/lpp/IBM/zoau/v1r3/lib/3.12'

LIBPATH: '/usr/lpp/IBM/zoau/v1r3/lib:/usr/lpp/IBM/cyp/v3r12/pyz/lib:/lib:/usr/lib:/usr/lpp/Printsrv/lib'

PATH: '/usr/lpp/IBM/zoau/v1r3/env/bin:/usr/lpp/IBM/zoau/v1r3/bin:/usr/lpp/IBM/cyp/v3r12/pyz/bin:/usr/bin:/usr/sbin:/usr/lpp/NFS:/usr/lpp/Printsrv/bin'

| Playbook

```
.ansible > collections > ansible_collections > ibm > ibm_zos_core > playbooks > ⓐ ds.yml
1  ---
2  - hosts: zsystem
3    collections:
4      - ibm.ibm_zos_core
5    gather_facts: no
6    vars:
7    environment: "{{ environment_vars }}"
8    tasks:
9      - name: Ping host - {{ ansible_host }}
10        ping:
11          register: result
12      - name: Response
```

Playbook

```
.ansible > collections > ansible_collections > ibm > ibm_zos_core > playbooks > ⓐ ds.yml
1  ---
2  - hosts: zsystem
3    collections:
4      - ibm.ibm_zos_core
5    gather_facts: no
6    vars:
7    environment: "{{ environment_vars }}"
8    tasks:
9      - name: Ping host - {{ ansible_host }}
10        ping:
11          register: result
12      - name: Response
```

```
~/ansible/collections/ansible_collections/ibm/ibm_zos_core/playbooks$ ansible-playbook -i inventory.ini ds.yml

PLAY [zsystem] *****

TASK [Ping host -] *****
ok: [zsystem]

TASK [Response] *****
ok: [zsystem] => {
  "msg": "pong"
}
```

Playbook

```
.ansible > collections > ansible_collections > ibm > ibm_zos_core > playbooks > ⓐ ds.yml
1  ---
2  - hosts: zsystem
3    collections:
4      - ibm.ibm_zos_core
5    gather_facts: no
6    vars:
7    environment: "{{ environment_vars }}"
8    tasks:
9      - name: Ping host - {{ ansible_host }}
10        ping:
11          register: result
12      - name: Response
```

```
~/ansible/collections/ansible_collections/ibm/ibm_zos_core/playbooks$ ansible-playbook -i inventory.ini ds.yml

PLAY [zsystem] *****

TASK [Ping host -] *****
ok: [zsystem]

TASK [Response] *****
ok: [zsystem] => {
  "msg": "pong"
}
```

Playbook

```
.ansible > collections > ansible_collections > ibm > ibm_zos_core > playbooks > ⓐ ds.yml
1  ---
2  - hosts: zsystem
3    collections:
4      - ibm.ibm_zos_core
5    gather_facts: no
6    vars:
7    environment: "{{ environment_vars }}"
8    tasks:
9      - name: Ping host - {{ ansible_host }}
10        ping:
11          register: result
12      - name: Response
```

```
~/ansible/collections/ansible_collections/ibm/ibm_zos_core/playbooks$ ansible-playbook -i inventory.ini ds.yml

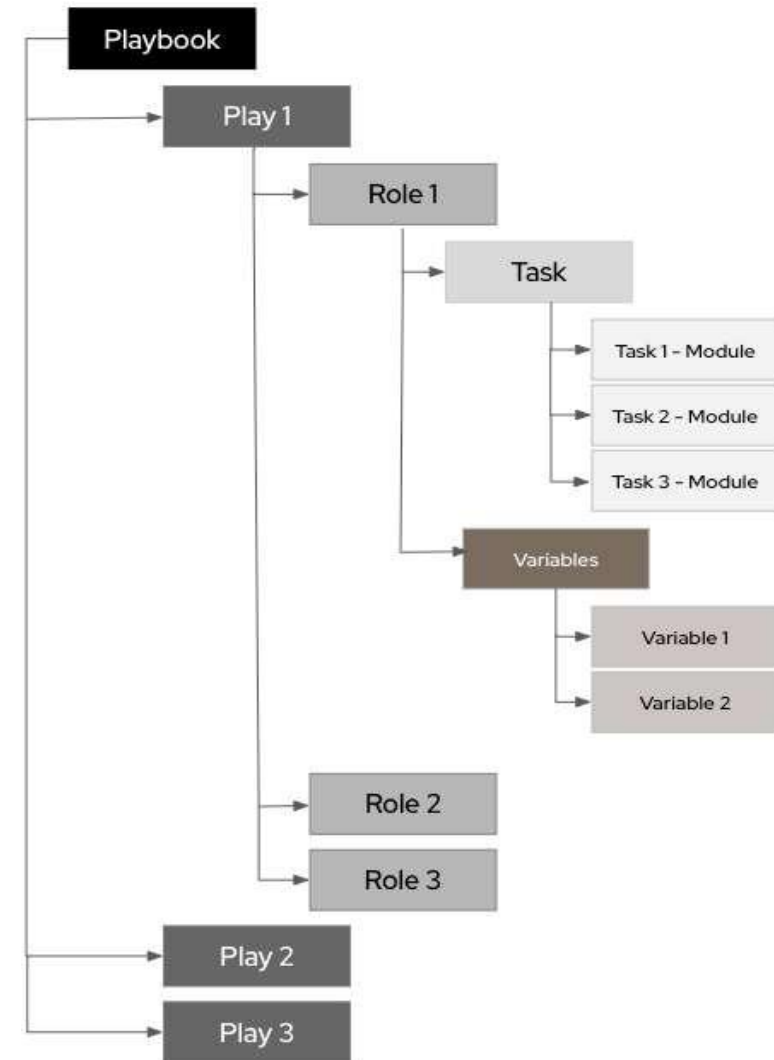
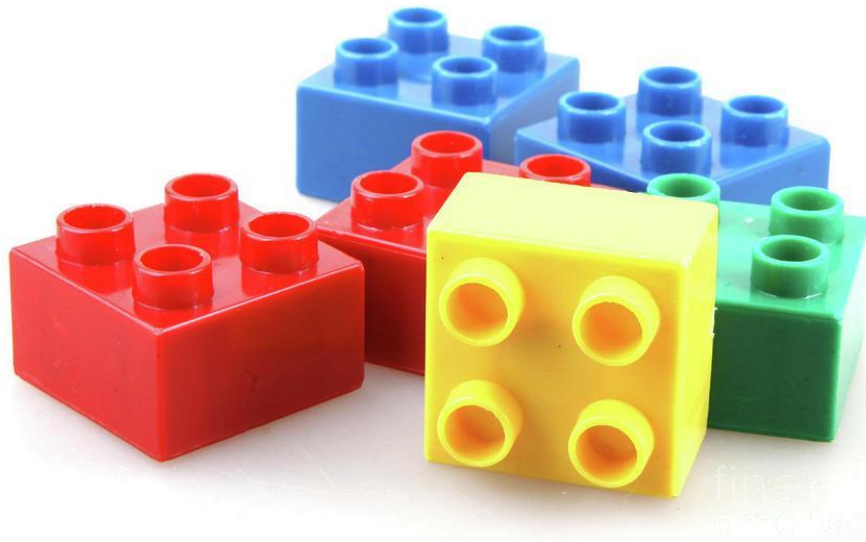
PLAY [zsystem] *****

TASK [Ping host -] *****
ok: [zsystem]

TASK [Response] *****
ok: [zsystem] => {
  "msg": "pong"
}
```

Automation Building Blocks

Robust, repeatable, and secure



OPS/MVS Modules

Open Source Extensions

Modules (6)



disable_rule

enable_rule

show_resource

show_rule

start_resource

stop_resource



Examples

```
- name: Enable MYRULE on ruleset OPSRULES on subsystem OPSS.  
  broadcom.ops.enable_rule:  
    ruleset: OPSRULES  
    rule: MYRULE  
    subsystem: OPSS
```



Scripts to Scale

Operator vs Ansible — Workflow Comparison

Reacting to a ticket manually at the console vs Ansible seeing the ticket and orchestrating the action

⚠️ OPERATOR CHALLENGES

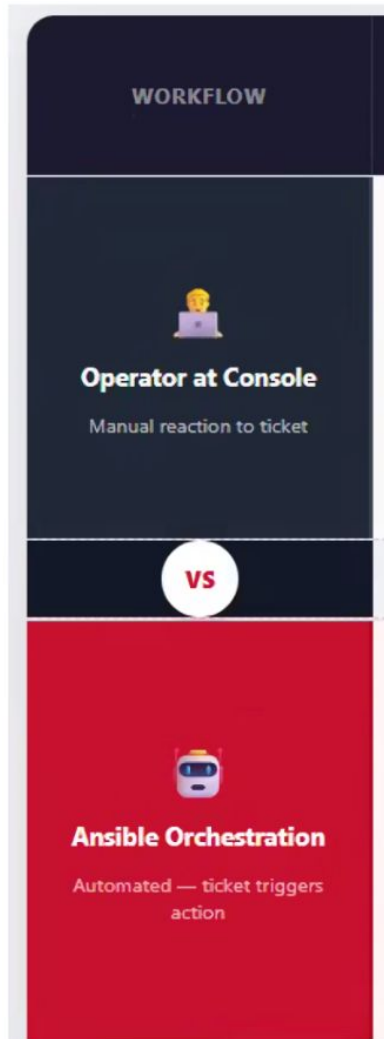
- 🕒 Delays tied to operator availability and shift schedules
- ⚠️ Inconsistent outcomes based on individual skill and experience
- 🗑️ Tribal knowledge disappears when experienced staff leave
- 🔄 Repetitive manual tasks consume valuable operator time
- 📖 Documentation is incomplete and varies person to person

✅ ANSIBLE ADVANTAGES

- ⚡ Responds to tickets instantly — 24 / 7 with no human intervention
- 🔄 Same playbook runs the same way every time — zero drift
- 📖 Knowledge lives in the playbook — not in someone's head
- 🚀 Frees operators to focus on higher-value, complex work
- 📝 Automatic logging delivers complete, consistent audit trails

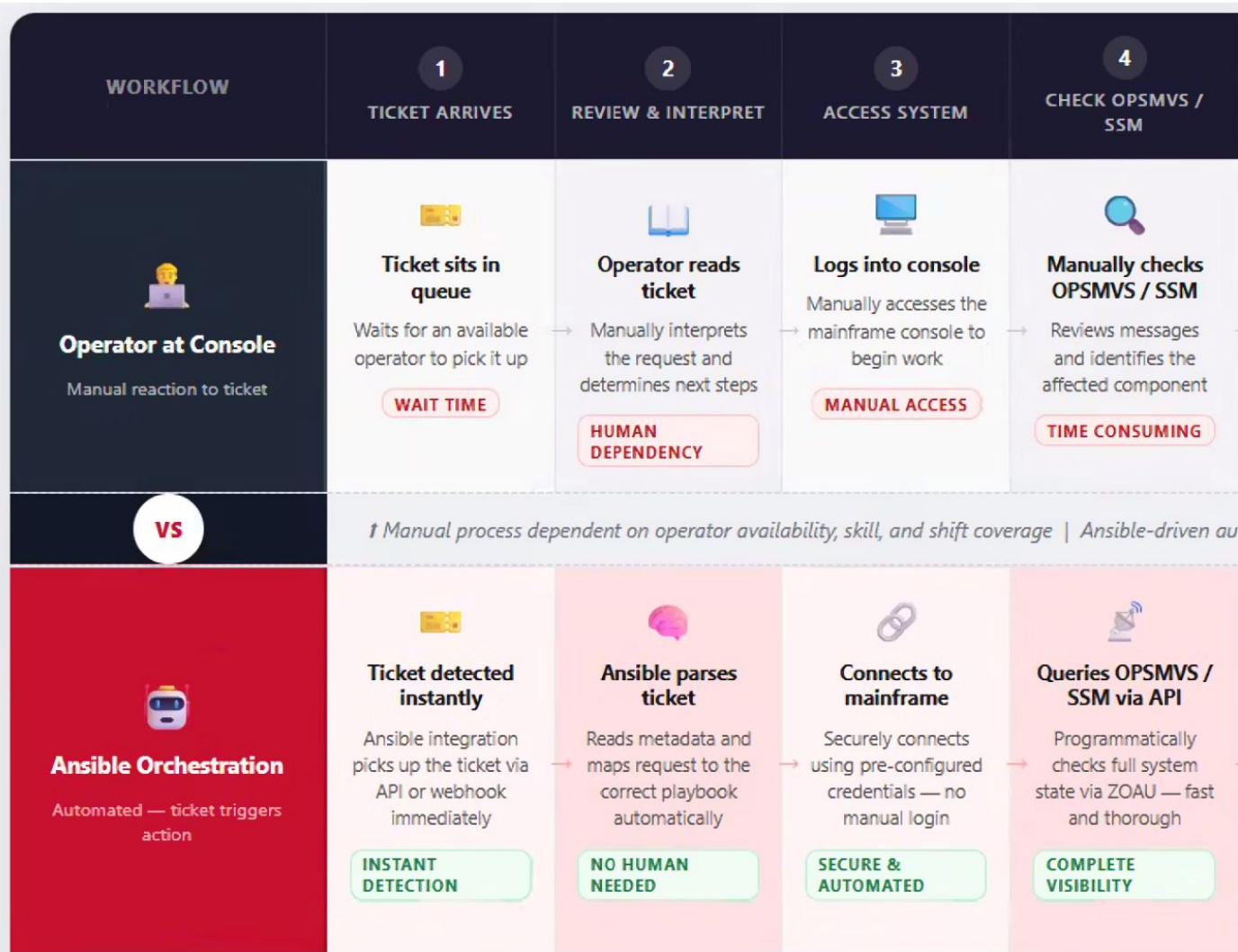
Operator vs. Ansible - Workflow Comparison

Reacting to a ticket manually vs. Orchestrating through OPS/MVS



Operator vs. Ansible - Workflow Comparison

Reacting to a ticket manually vs. Orchestrating through OPS/MVS



Operator vs. Ansible - Workflow Comparison

Reacting to a ticket manually vs. Orchestrating through OPS/MVS

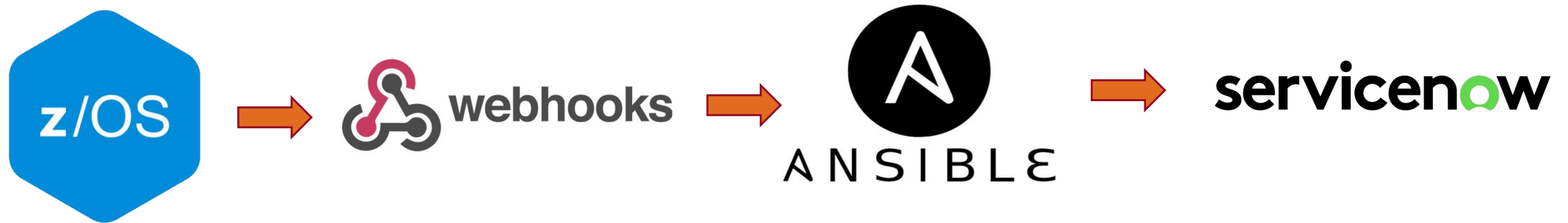
WORKFLOW	1 TICKET ARRIVES	2 REVIEW & INTERPRET	3 ACCESS SYSTEM	4 CHECK OPSMVS / SSM	5 REFERENCE STEPS	6 EXECUTE ACTION	7 VALIDATE OUTCOME	8 CLOSE TICKET
 Operator at Console Manual reaction to ticket	 Ticket sits in queue Waits for an available operator to pick it up WAIT TIME	 Operator reads ticket Manually interprets the request and determines next steps HUMAN DEPENDENCY	 Logs into console Manually accesses the mainframe console to begin work MANUAL ACCESS	 Manually checks OPSMVS / SSM Reviews messages and identifies the affected component TIME CONSUMING	 Looks up runbook Relies on documented steps or tribal knowledge to proceed INCONSISTENCY RISK	 Types commands manually Executes steps one by one at the console ERROR PRONE	 Spot-checks result Manually reviews output to verify the action completed SPOT CHECK ONLY	 Manually updates ticket Writes notes and closes — quality varies by person INCONSISTENT RECORDS
VS	↑ Manual process dependent on operator availability, skill, and shift coverage Ansible-driven automation below triggers instantly on ticket creation ↓							
 Ansible Orchestration Automated — ticket triggers action	 Ticket detected instantly Ansible integration picks up the ticket via API or webhook immediately INSTANT DETECTION	 Ansible parses ticket Reads metadata and maps request to the correct playbook automatically NO HUMAN NEEDED	 Connects to mainframe Securely connects using pre-configured credentials — no manual login SECURE & AUTOMATED	 Queries OPSMVS / SSM via API Programmatically checks full system state via ZOAU — fast and thorough COMPLETE VISIBILITY	 Selects correct playbook Pre-validated steps run identically every time — no tribal knowledge needed ALWAYS CONSISTENT	 Executes automatically Ansible runs commands and applies changes — fast, precise, repeatable ZERO MANUAL EFFORT	 Runs full post-validation Playbook confirms success — alerts triggered immediately if anything fails FULL VALIDATION	 Auto-closes with full logs Detailed execution logs written back to ticket — complete audit trail every time COMPLETE AUDIT TRAIL



Forward Looking

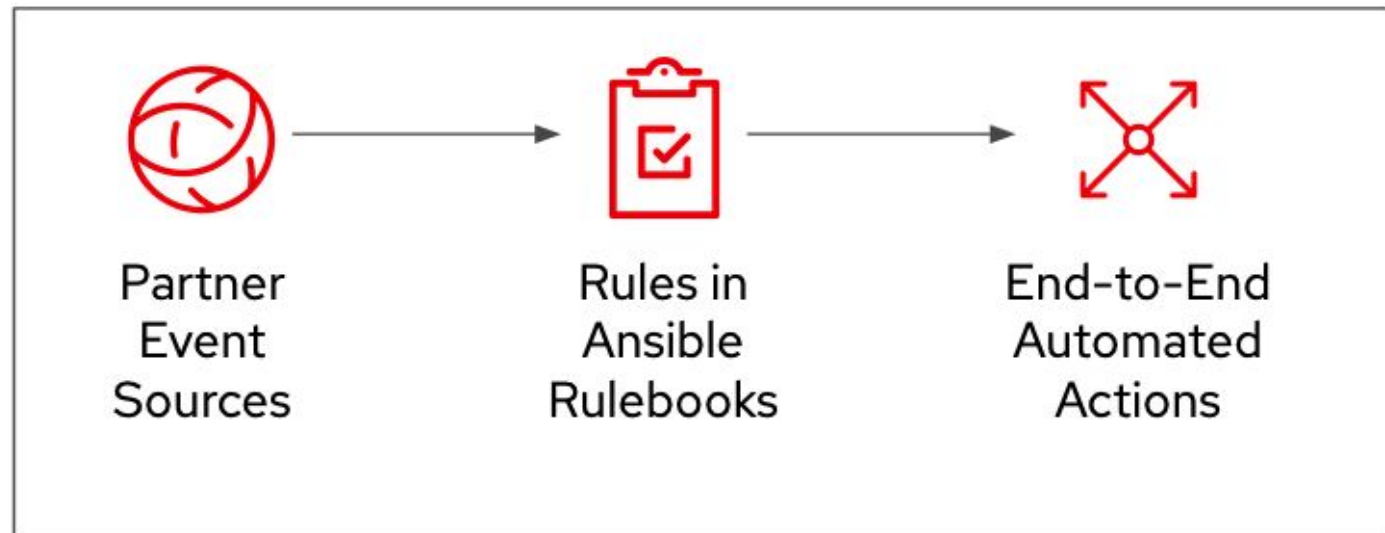
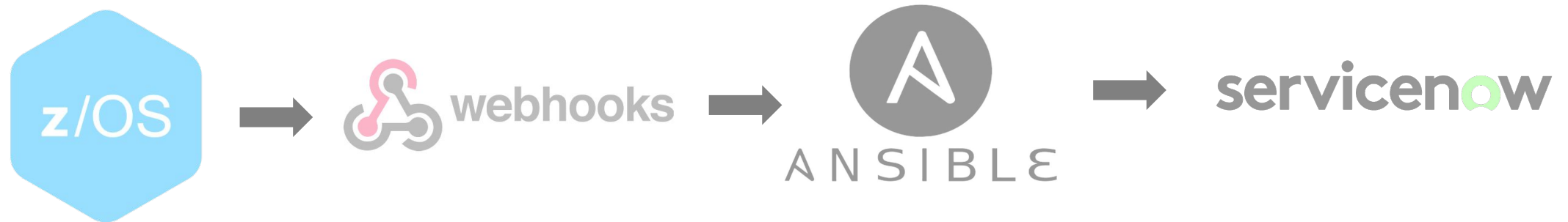
| Integrations & Forward Looking

Event Driven Automation



Integrations & Forward Looking

Event Driven Automation



Hands-On Workshop: Ethical Mainframe Hacking (BYOD)

Join us for a **hands-on** opportunity to learn about ethical hacking and z/OS!

- Explore a vulnerable web application with a database backend,
- Learn the process of identification, fuzzing, iterating potential exploits
- Ultimately hacking into the mainframe's database



Location: **Room 409**

- Monday Aug 17 2:30-3:30pm
- Wednesday Aug 19 9:15-10:15am

Limited space — come early!



Your feedback is important!

Submit a session evaluation for each session you attend:

www.share.org/evaluation

