ANSIBLE AUTOMATES

Automate Windows Environments with Ansible

Orcun Atakan (oatakan@redhat.com) - Sr. Solution Architect, Red Hat
What we’ll be discussing...

- What is Ansible?
- Windows Management with Ansible
- What is DSC?
- Why Use Ansible with DSC?
- Demo: Rolling update of a .NET application on Windows
42,000+ Stars on GitHub
3500+ Ansible modules
500,000+ Downloads a month
Why Ansible? (for Windows)

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

**POWERFUL**
- Enable many use cases
- Works hand-in-hand with DSC resources
- Easy platform enablement
- Leverage Powershell

**AGENTLESS**
- Ideal for Windows remoting
- No agents to exploit or update
- Standards-based WinRM
ANSIBLE AUTOMATION WORKS ACROSS TEAMS

BUSINESS
DEV/QA
COMPUTE
NETWORK/SECURITY
I.T. OPERATIONS
WINDOWS AUTOMATION

100+ Windows Modules

1,300+ Powershell DSC resources

ansible.com/windows
WHAT CAN I DO USING ANSIBLE FOR WINDOWS

Native Windows support uses PowerShell remoting to manage Windows in the same Ansible agentless way.

- Install and uninstall MSIs
- Gather facts on Windows hosts
- Enable and disable Windows features
- Start, stop, and manage Windows Services
- Create and Manage local users and groups
- Manage Windows packages via Chocolatey package manager
- Manage and install Windows updates
- Fetch files from remote sites
- Push and execute any Powershell scripts
- name: start IIS/stop firewall
  hosts: windows-web
  become: yes
  tasks:
  - name: IIS is running
    win_service:
      name: W3Svc
      state: running

  - name: firewall service is stopped/disabled
    win_service:
      name: MpsSvc
      state: stopped
      start_mode: disabled
---
- name: windows playbook
  hosts: new_servers

  tasks:
  - name: ensure local admin account exists
    win_user:
      name: localadmin
      password: '{{ local_admin_password }}'
      groups: Administrators
AUTOMATION FOR EVERYONE: WINDOWS USERS

---

- **name**: Created AD user account

  ```
  win_domain_user:
    name: '{{ win_user_name }}'
    firstname: '{{ win_user_firstname }}'
    surname: '{{ win_user_surname }}'
    password: '{{ win_user_password }}'
    groups: '{{ win_user_groups }}'
    email: '{{ win_user_email }}'
    state: present
  ```
AUTOMATION FOR EVERYONE: WINDOWS USERS

<table>
<thead>
<tr>
<th>NEW AD USER</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INVENTORY SURVEY PREVIEW</td>
</tr>
<tr>
<td><strong>FIRST NAME</strong></td>
<td>john</td>
</tr>
<tr>
<td><strong>SURNAME</strong></td>
<td>Smith</td>
</tr>
<tr>
<td><strong>PASSWORD</strong></td>
<td>Will validate against company password policies</td>
</tr>
<tr>
<td>SHOW</td>
<td>....................</td>
</tr>
<tr>
<td><strong>PREFERRED EMAIL ADDRESS</strong></td>
<td>First Initial last <a href="mailto:name@example.com">name@example.com</a></td>
</tr>
<tr>
<td><a href="mailto:jsmith@example.com">jsmith@example.com</a></td>
<td></td>
</tr>
<tr>
<td><strong>ENTER DOMAIN GROUPS</strong></td>
<td>THIS ACCOUNT SHOULD BE ADDED TO</td>
</tr>
<tr>
<td>Must match AD exactly</td>
<td></td>
</tr>
<tr>
<td>admin</td>
<td>human_resources</td>
</tr>
<tr>
<td><strong>MANAGER TO EMAIL AFTER COMPLETION</strong></td>
<td><a href="mailto:bigboss@example.com">bigboss@example.com</a></td>
</tr>
</tbody>
</table>

CANCEL | NEXT
- name: ensure common OS updates are current
  win_updates:
    register: update_result

- name: reboot and wait for host if updates change require it
  win_reboot:
    when: update_result.reboot_required
AUTOMATION FOR EVERYONE: WINDOWS UPDATES (COMPLEX)

---

- **name**: ensure common OS updates are current
  
  `win_updates`:
  
  category_names: "{{ categories }}"

  blacklist: "{{ blacklist_package | default(omit, true) }}"

  whitelist: "{{ whitelist_package | default(omit, true) }}"

  register: update_result

- **name**: reboot and wait for host if updates change require it
  
  `win_reboot`:

  when: update_result.reboot_required
AUTOMATION FOR EVERYONE: WINDOWS UPDATES (COMPLEX)
## AUTOMATION FOR EVERYONE: WINDOWS UPDATES (COMPLEX)

<table>
<thead>
<tr>
<th>Windows VM</th>
<th>List of Required Updates/Packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.128.107</td>
<td>• Definition Update for Windows Defender Antivirus - KB2267602 (Definition 1.297.948.0)</td>
</tr>
<tr>
<td>192.168.128.119</td>
<td>• Windows Malicious Software Removal Tool x64 - July 2019 (KB890830)</td>
</tr>
<tr>
<td></td>
<td>• Microsoft Silverlight (KB4481252)</td>
</tr>
<tr>
<td></td>
<td>• 2019-07 Cumulative Update for .NET Framework 3.5, 4.7.2, 4.8 for Windows 10 Version 1809 for x64 (KB4507419)</td>
</tr>
<tr>
<td></td>
<td>• 2019-07 Cumulative Update for Windows 10 Version 1809 for x64-based Systems (KB4507469)</td>
</tr>
<tr>
<td>192.168.150.31</td>
<td>• Windows Malicious Software Removal Tool x64 - July 2019 (KB890830)</td>
</tr>
<tr>
<td></td>
<td>• 2019-07 Cumulative Update for Windows Server 2016 for x64-based Systems (KB4507460)</td>
</tr>
<tr>
<td></td>
<td>• 2019-07 Servicing Stack Update for Windows Server 2016 for x64-based Systems (KB4509091)</td>
</tr>
</tbody>
</table>
AUTOMATION FOR EVERYONE: DOMAIN MEMBERSHIP

---

- name: update domain and reboot
  hosts: windows_servers
  tasks:
    - name: ensure domain membership
      win_domain_membership:
        dns_domain_name: contoso.corp
        domain_admin_user: '{{ domain_admin_username }}'
        domain_admin_password: '{{ domain_admin_password }}'
        state: domain
        register: domain_result

    - name: reboot and wait for host if domain change require it
      win_reboot:
      when: domain_result.reboot_required
What is DSC?

> Windows Management Platform built in
  ● Ships natively with Windows Server 2012 R2 and Windows 8.1 and newer
  ● Requires PowerShell v4 or greater

> Configuration based declarative model
  ● Define desired state in configuration
  ● DSC determines how to execute on target

> Push or Pull Architecture
Why Use DSC with Ansible?

- Both declarative & end-state oriented
- Compliment each other
- Rich community ecosystem for both
- Extend end-to-end use cases beyond Windows management
- Scale using Ansible lightweight architecture
- Ansible Tower provides enterprise capabilities managing Windows
Where to use Ansible Windows Modules vs DSC resources?

Reasons for using an Ansible module over a DSC resource:

- The host does not support PowerShell v5.0, or it cannot easily be upgraded
- The DSC resource does not offer a feature present in an Ansible module
- DSC resources have limited check mode support, while some Ansible modules have better checks
- DSC resources do not support diff mode, while some Ansible modules do
- Custom resources require further installation steps to be run on the host beforehand, while Ansible modules are in built-in to Ansible

Reasons for using a DSC resource over an Ansible module:

- The Ansible module does not support a feature present in a DSC resource
- There is no Ansible module available
<table>
<thead>
<tr>
<th>Playbook Step</th>
<th>Ansible Configuration</th>
</tr>
</thead>
</table>
| - name: Install IIS Web-Server  
**win_feature:**  
  name: Web-Server  
  state: present  
  restart: True  
  include_sub_features: True  
  include_management_tools: True |
| - name: Create IIS site  
**win_iis_website:**  
  name: Ansible  
  state: started  
  physical_path: c:sites\Ansible |
| - name: Add HTTP webbinding to IIS  
**win_iis_webbinding:**  
  name: Ansible  
  protocol: http  
  port: 8080  
  ip: '*'  
  state: present |
| - name: Install required DSC module  
**win_psmodule:**  
  name: xWebAdministration  
  state: present |
| - name: Install IIS Web-Server  
**win_dsc:**  
  resource_name: windowsfeature  
  name: Web-Server |
| - name: Create IIS site  
**win_dsc:**  
  resource_name: xWebsite  
  Ensure: Present  
  Name: Ansible  
  State: Started  
  PhysicalPath: c:sites\Ansible  
  BindingInfo:  
    - Protocol: http  
    Port: 8080  
    IPAddress: '*' |

Example playbooks with Ansible Modules vs DSC resources
**Use win_dsc module vs Powershell**

<table>
<thead>
<tr>
<th>- name: Install required DSC module</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>win_psmodule:</strong></td>
</tr>
<tr>
<td>name: xWebAdministration</td>
</tr>
<tr>
<td>state: present</td>
</tr>
</tbody>
</table>

- name: Install IIS Web-Server

**win_dsc:**

- resource_name: windowsfeature
  name: Web-Server

- name: Create IIS site

**win_dsc:**

- resource_name: xWebsite
  Ensure: Present
  Name: Ansible
  State: Started
  PhysicalPath: c:\sites\Ansible
  BindingInfo:
    - Protocol: http
      Port: 8080
      IPAddress: '*'

# Import the module
Import-DscResource -Module xWebAdministration, PSDesiredStateConfiguration

Node $NodeName

{  
  # Install the IIS role
  WindowsFeature IIS
  {  
    Ensure = 'Present'
    Name = 'Web-Server'
  }

  xWebsite DefaultSite
  {  
    Ensure = 'Present'
    Name = 'Ansible'
    State = 'Started'
    PhysicalPath = 'c:\sites\Ansible'
    DependsOn = '[WindowsFeature]IIS'
    BindingInfo = MSFT_xWebBindingInformation
    {  
      Protocol = 'http'
      Port = '8080'
      IPAddress = '*'
    }
  }
}
Handle Credentials with win_dsc Module

- By default win_dsc module uses SYSTEM account

- You can use PsDscRunAsCredential attribute to run as another user:

```yaml
- name: use win_dsc with PsDscRunAsCredential to run as a different user
  win_dsc:
    resource_name: Registry
    Ensure: Present
    Key: HKEY_CURRENT_USER\ExampleKey
    ValueName: TestValue
    ValueData: TestData
    PsDscRunAsCredential_username: '{{ ansible_user }}'
    PsDscRunAsCredential_password: '{{ ansible_password }}'
    no_log: true
```
Some Example DSC Resources

- **Built-in:**
  - Archive
  - File
  - Group
  - Package
  - WindowsFeature
  - And more..

- **Custom resources provided by Microsoft and the community:**
  - Domain Controller
  - IIS Web Site
  - SQL Server Cluster
  - Failover Cluster
  - DNS
  - And many more..
Demo - .Net Music Store App with DB

- VMware - Test
  - Win2016 + IIS
  - Win2016 + SQL DB

- Azure - Dev
  - Win2016 + IIS
  - Win2016 + SQL DB

- AWS - Prod
  - Load Balancer
  - Win2016 + IIS
  - Win2016 + SQL DB
Your applications and systems are more than just collections of configurations. They're a finely tuned and ordered list of tasks and processes that result in your working application.

Ansible can do it all:
- Provisioning
- App Deployment
- Configuration Management
- Multi-tier Orchestration
Demo Time !
Windows is a first class citizen within the Ansible ecosystem!
How to obtain a list of Ansible Windows Modules and DSC Resources

- Ansible Modules:
  - https://docs.ansible.com/ansible/latest/modules/list_of_windows_modules.html

- Built-in DSC Resources:
  - https://docs.microsoft.com/en-us/powershell/dsc/builtinresource
  - Or run this powershell command: Find-DscResource

- DSC Resources on Github:
  - https://github.com/PowerShell/DscResources

- DSC Resources on Powershell Gallery:
  - https://www.powershellgallery.com
ANSIBLE AUTOMATES

THANK YOU

youtube.com/AnsibleAutomation  facebook.com/ansibleautomation
linkedin.com/company/Red-Hat  twitter.com/ansible