LEVVEL

Ansible Tower AWS Windows Application Deployment

Using Ansible Tower to Deploy a Windows Server AWS Instance, and Install and Manage an Application

2019

Introduction

Ansible and Ansible Tower are known to be great automation and configuration management tools, and are enjoyed by developers and IT operations departments alike. Because the majority of environments with Ansible Tower are primarily comprised of Linux servers, I often receive questions about how well Ansible Tower works with Windows servers. To the surprise of some people, Ansible Tower is a great product for any IT environment! Teams can deploy and manage windows servers quite easily, as Ansible Tower by default comes with modules that can be used to deploy and manage Windows. The following content gives some information and use cases on how to use Ansible Tower with your Windows environment in AWS.

Ansible Tower & Windows Server

As with Linux, almost anything can be scripted and automated in Windows. Powershell is a very powerful tool that every savvy windows server administrator should know. In order for Ansible to manage Windows servers, WinRM and PSRemoting must be enabled.

Ansible and Ansible Tower uses the WinRM (Windows Remote Management) feature to execute commands, including Powershell scripts remotely. The PyWinRM python module that must be installed on the Ansible host provides the libraries to query the WinRM daemon.

Use Cases

Ansible provides a comprehensive solution to automating your entire environment, with an extensive list of pre-built modules. For example, there are modules for Cisco devices, NetApp, Servers (Linux & Windows), cryptography, etc. What this means for you and your environment is that you now have the capability to automate everything using a stable and high performance automation engine.

The specific scenario I will describe in this post, is a simple end-to-end deployment of an IIS application in a Windows server. Ansible will create and launch and AWS EC2 instance using an AMI, enable WinRM and PSRemoting, set the administrator credentials, install all of the required windows server features, and then deploy and test an application.

Prereqs

Several things will be needed for this exercise.

1. An AWS account with a secret key id and secret key to access the API
2. An Ansible Tower server with version 3.3.0
3. SSH Access to the Ansible Tower Server

Preparing Ansible Tower

The first thing we need to do is prepare our Ansible Tower server. This blog post assumes you already have a base installation of Ansible Tower on a Red Hat Enterprise Linux server. I'm currently using Ansible Tower 3.3.0.

Install Required Packages

1. In order to use the WinRM protocol on Windows servers, you need to install the 'pywinrm' python module. Python requires the 'boto' modules to access the AWS API. Pip should be present to install the required modules.

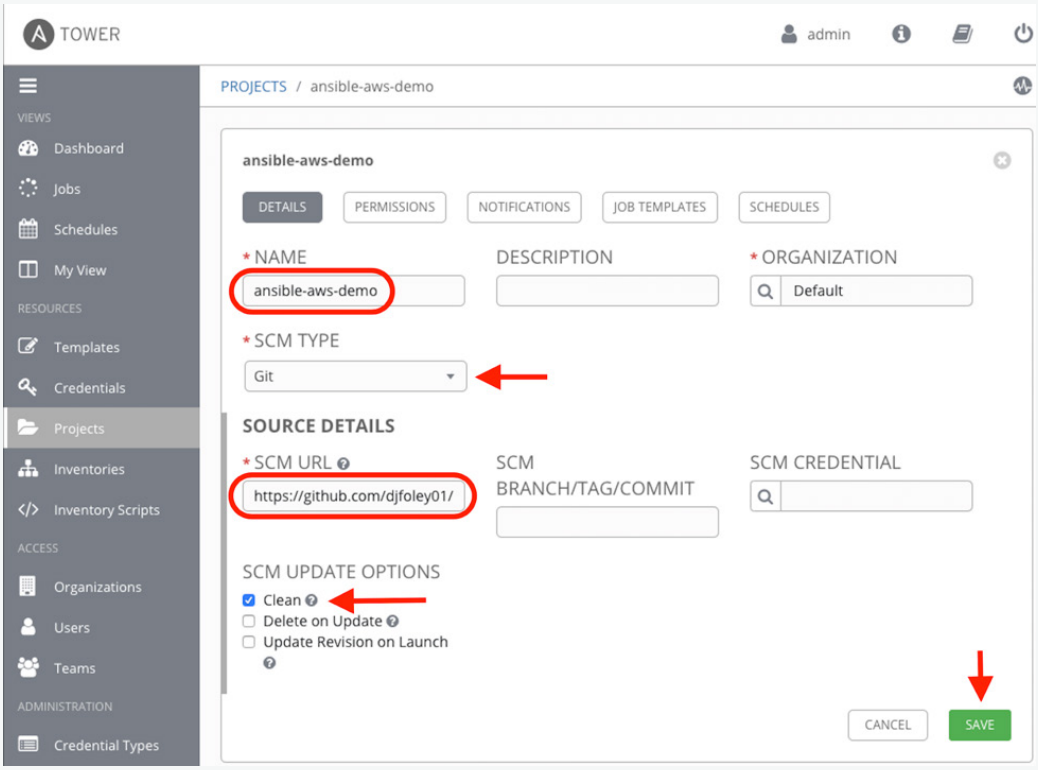
```
[root@ip-172-30-0-141 ~]# tower-manage --version
3.3.0
[root@ip-172-30-0-141 ~]# python --version
Python 2.7.5
[root@ip-172-30-0-141 ~]# yum install python2-pip
[root@ip-172-30-0-141 ~]# pip install pywinrm python2-boto python2-boto3
python2-botocore
```

2. Once you have the 'pywinrm' module and the python2 'boto' modules installed, Ansible Tower will also need some preparation.

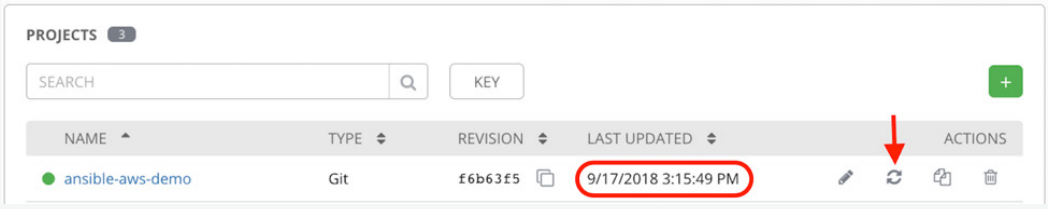
Create Project

A Project in Ansible Tower defines the location of the playbooks you wish to use in your template. The template will be the last object in Ansible Tower we will create.

1. From the Dashboard, click on ‘Projects’ on the left hand side of the screen, click on the ‘+’ button. Enter the ‘NAME’ for the project (in this case I chose ‘ansible-aws-demo’), and the ‘SCM Type’ should be ‘Git’ (the ‘SCM URL’ I used is ‘https://github.com/djfoley01/aws-ansible-demo’ where I have stored the playbook for this demo). Check the box for ‘Clean’, then click ‘SAVE’.



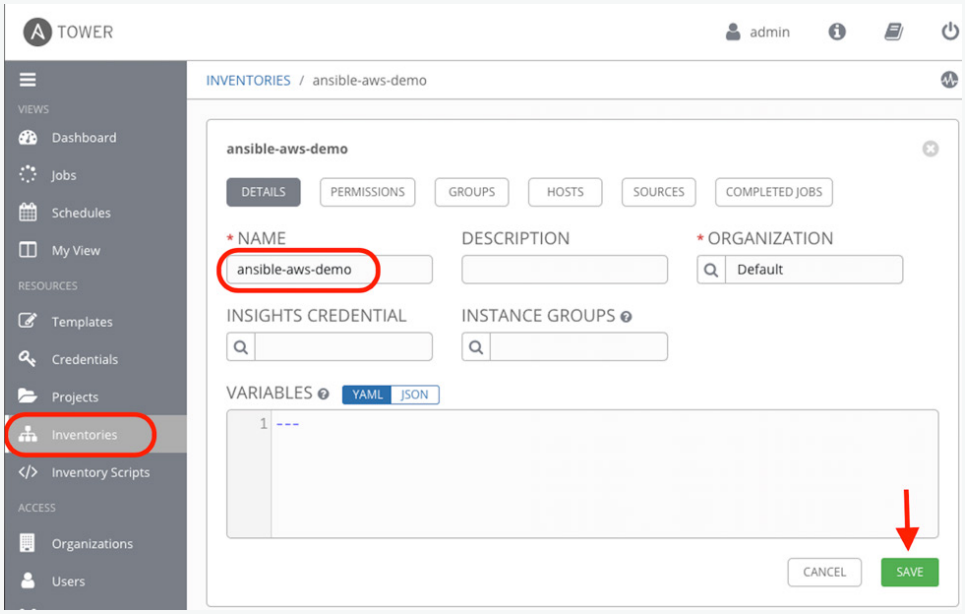
2. Confirm the project has synchronized the playbooks from the SCM URL. If you do not see a ‘LAST UPDATED’ date, click on sync button.



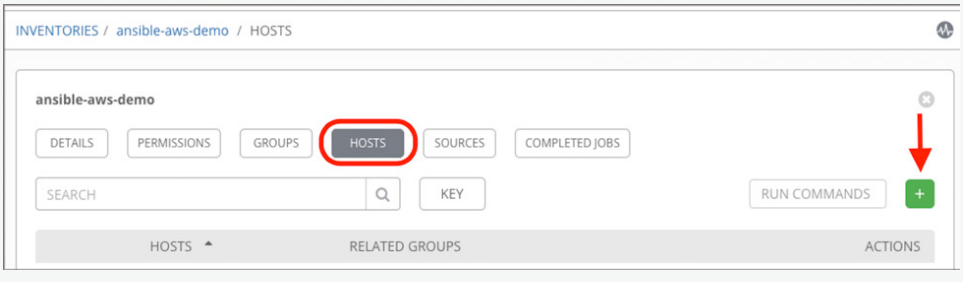
Create Inventory

1. Create an inventory. This inventory will only include ‘localhost’ as we will be primarily accessing the AWS API to launch a Windows instance. Once the Windows instance is deployed, Ansible Tower will dynamically add the newly deployed Windows instance to the inventory for the second role (the Windows execution tasks).

    » Click on ‘Inventory’ on the left hand side of the screen, then on the ‘+’ button, provide a name, and click save.

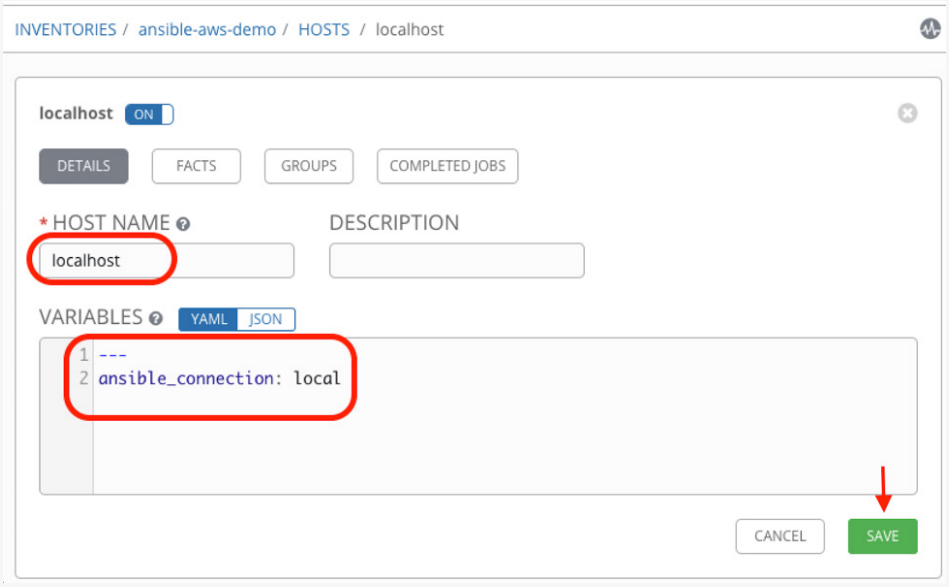


    » Click on ‘HOSTS’, and then click on the ‘+’ button to add a new host.

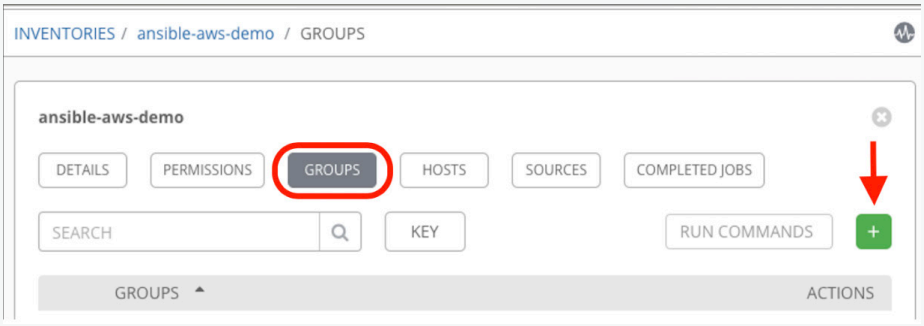


    » Enter the hostname ‘localhost’, add the ansible_connection type under variables, then click save.

```
---
ansible_connection: local
```

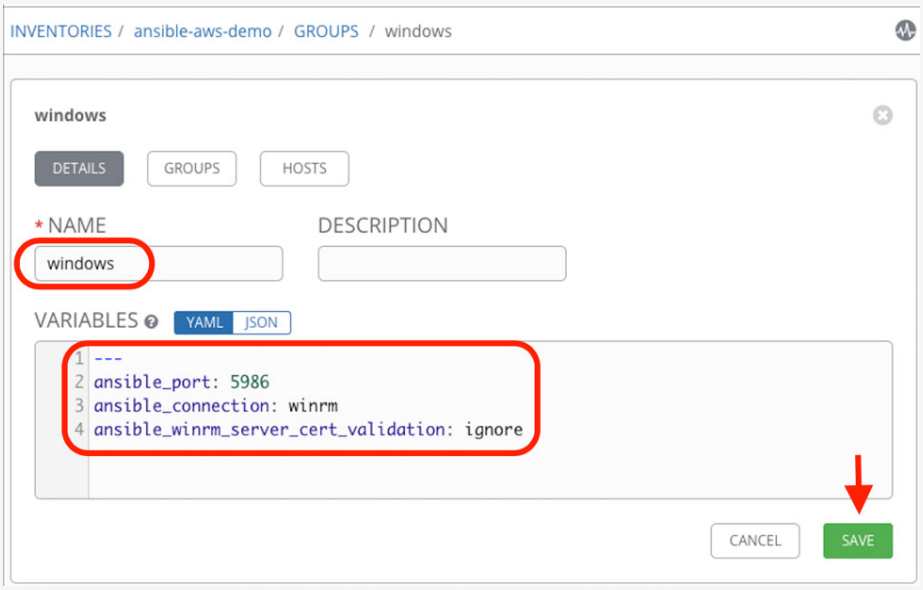


» Click on ‘GROUPS’, then click the ‘+’ button.



» Insert the name ‘windows’. Using the ‘windows’ name is critical because that is how it is called in the playbook. After adding the name, add the variables required to connect to the Windows machines using WinRM.

```
---
ansible_port: 5986
ansible_connection: winrm
ansible_winrm_server_cert_validation: ignore
```

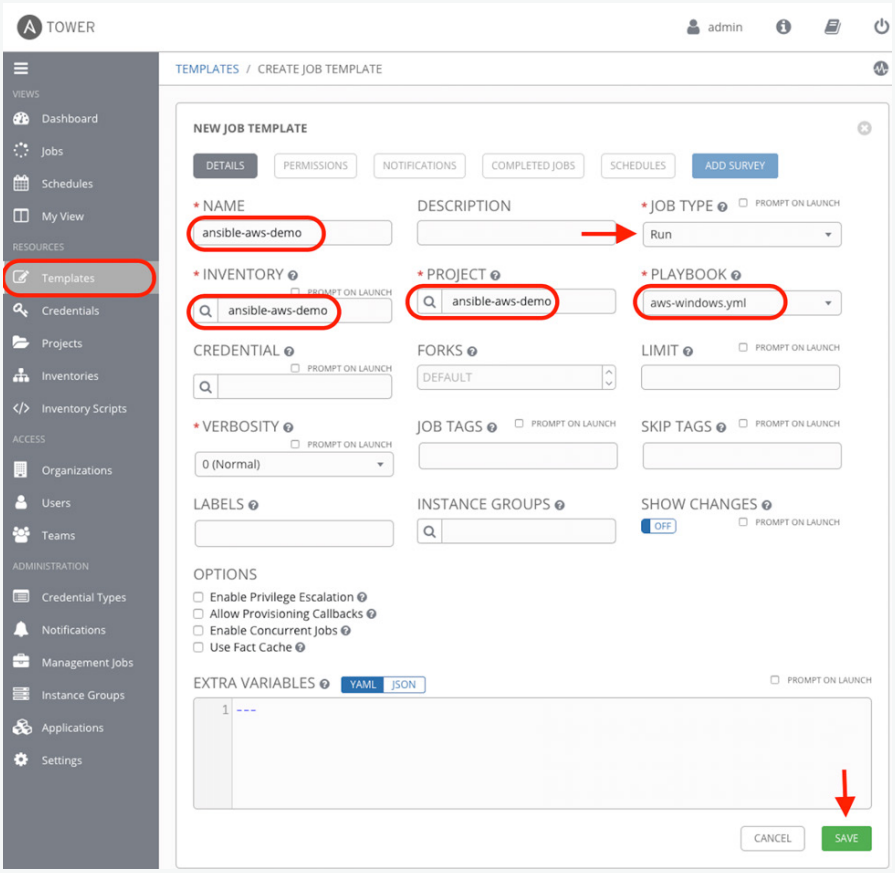


After you have completed adding the group, the inventory is complete. Next step is to create the template.

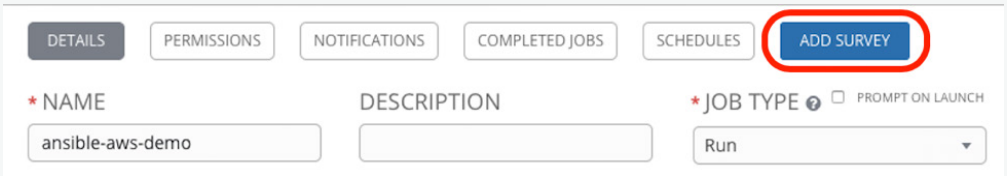
Create Template

The template is specifying all of the parameters needed to execute the job. We will be creating a survey — a pop-up prior to the job executing requesting you to fill out a list of variables. There are different types of data fields, including lists, text areas, and password fields. We will be using these three different field types so you can see the differences in how data is handled in Ansible Tower. Password fields, for example, are encrypted, and you can never actually see the data that has been entered; it will only show as '$encrypted$'.

1. Click on 'Templates' on the left hand side of the screen, then click on the '+' button to create a new template. Type in the NAME (for this demo I used 'ansible-aws-demo'), and the JOB TYPE needs to be 'Run'. Use the inventory we created in the step prior 'ansible-aws-demo', use the project created in the first step 'ansible-aws-demo', then select the playbook that will call the roles in the playbook 'aws-windows.yml'. Roles will be discussed in a later section.



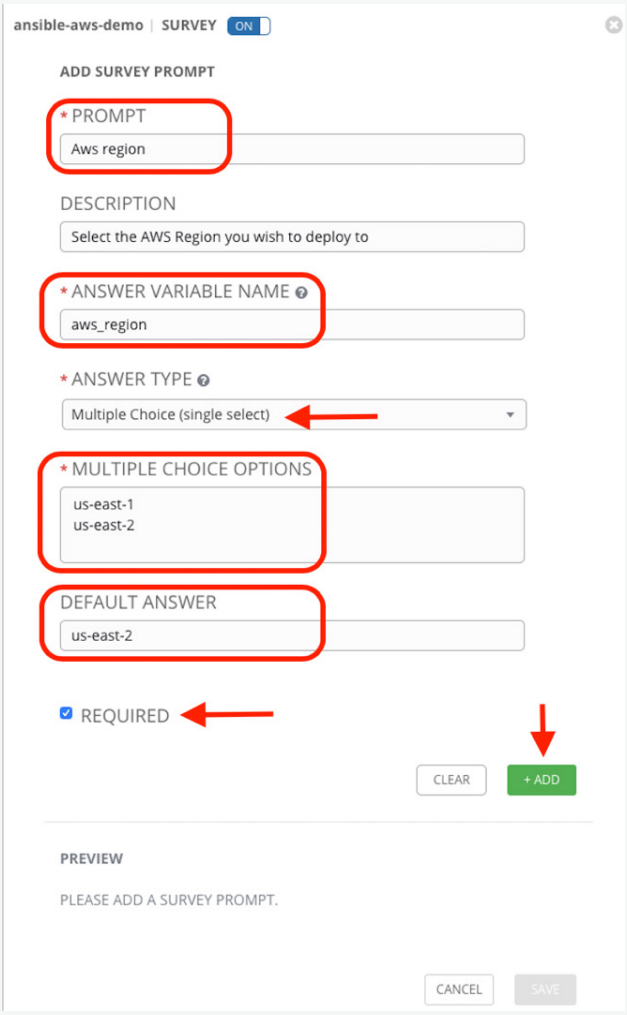
2. After you click save, the ‘ADD SURVEY’ button will become dark blue and you will be able to click on it.



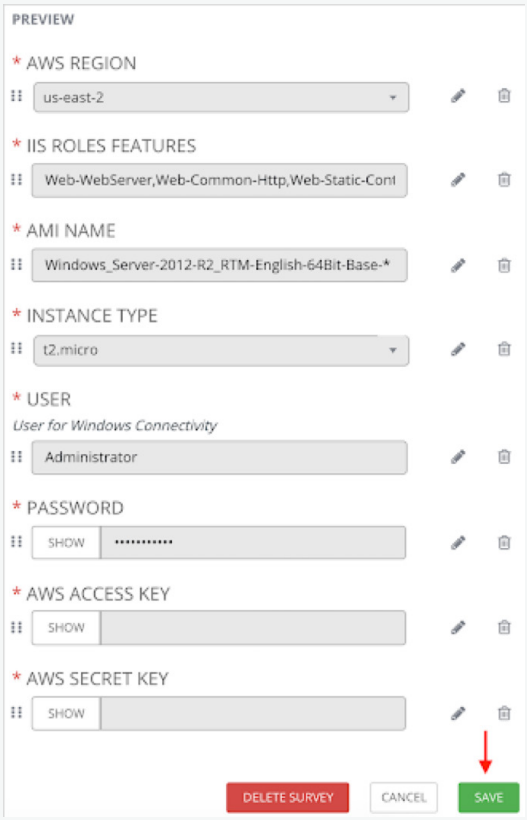
3. In the survey, we will need to create a multiple fields to fulfill several variables (seen below). These variable names need to be used as-is, since they are called in the playbook.

Variable Name	Default Value	Answer Type
aws_region	us-east-2	Multiple-Choice (Single Select)
iis_role_features	Web-WebServer,Web-Common-Http,Web-Static-Content,Web-Default-Doc,Web-Http-Errors,Web-Http-Redirect,Web-ASP,Web-Http-Logging,AS-Web-Support	Text Area
ami_name	Windows_Server-2012-R2_RTM-English-64Bit-Base-*	Text Area
instance_type	t2.micro	Multiple-Choice (Single Select)
ansible_user	Administrator	Text Area
ansible_password	<Admin Password>	Password
aws_access_key_id	DO NOT SET	Password
aws_secret_access_key	DO NOT SET	Password

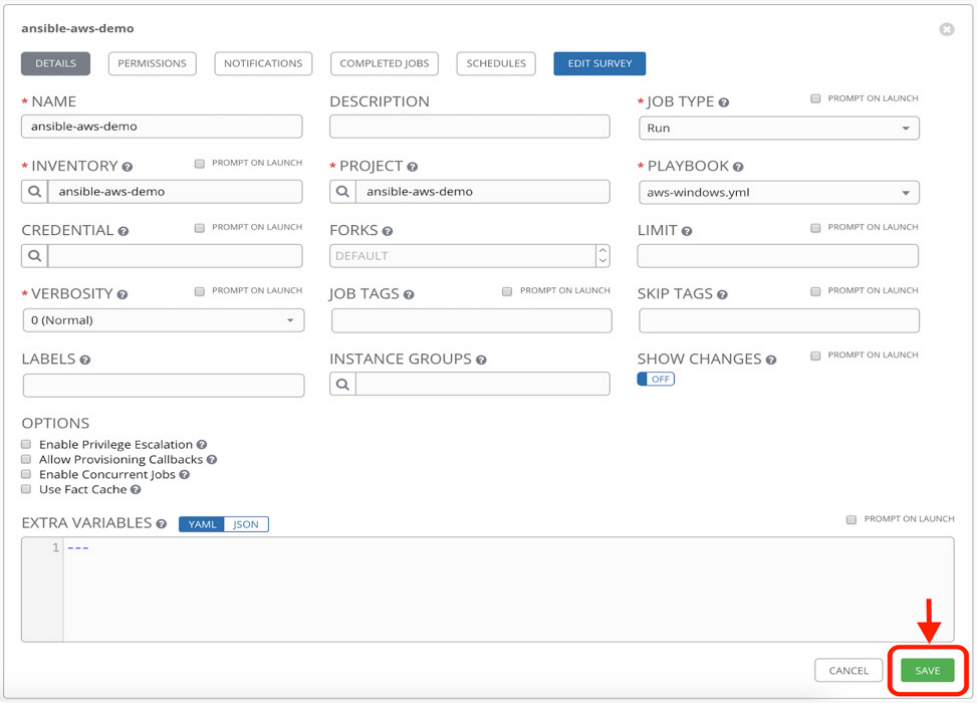
4. After you click 'ADD SURVEY', fill out all fields shown in the example below. The first field is the PROMPT, essentially the display name for the variable in the survey. ANSWER VARIABLE NAME is the actual name of the variable and how it will be called by the playbook. The ANSWER TYPE identifies the data type. In the example below, I created a single select multiple choice type. This allows me to create predefined MULTIPLE CHOICE OPTIONS: us-east-1, us-east-2. In the DEFAULT ANSWER field I put us-east-2 simply because that is where I want to deploy the majority of my test instances. Make the prompt field REQUIRES, then click '+ADD'.



5. Once you have clicked '+ADD', you will see your new survey prompt field down at the bottom of the screen under PREVIEW, along with any default value you may have set.
6. Go through the list and add all of the variables from the list above. When you have finished adding the variable prompts, your PREVIEW section should look like the following.

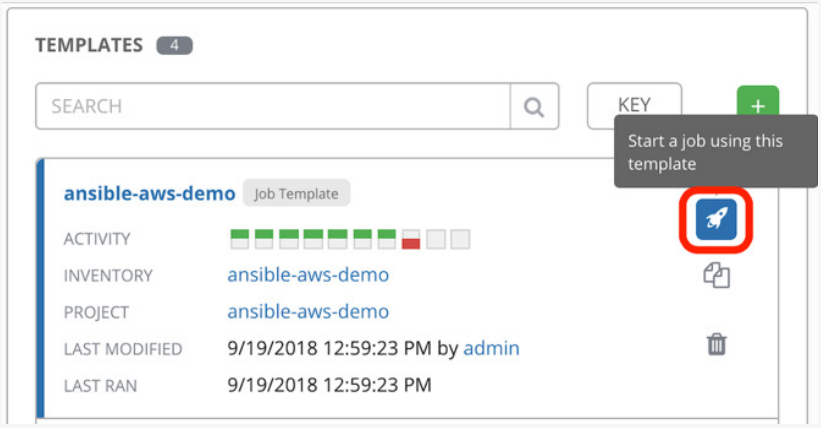


7. After you have verified that the preview looks correct, click ‘SAVE’ for your survey.

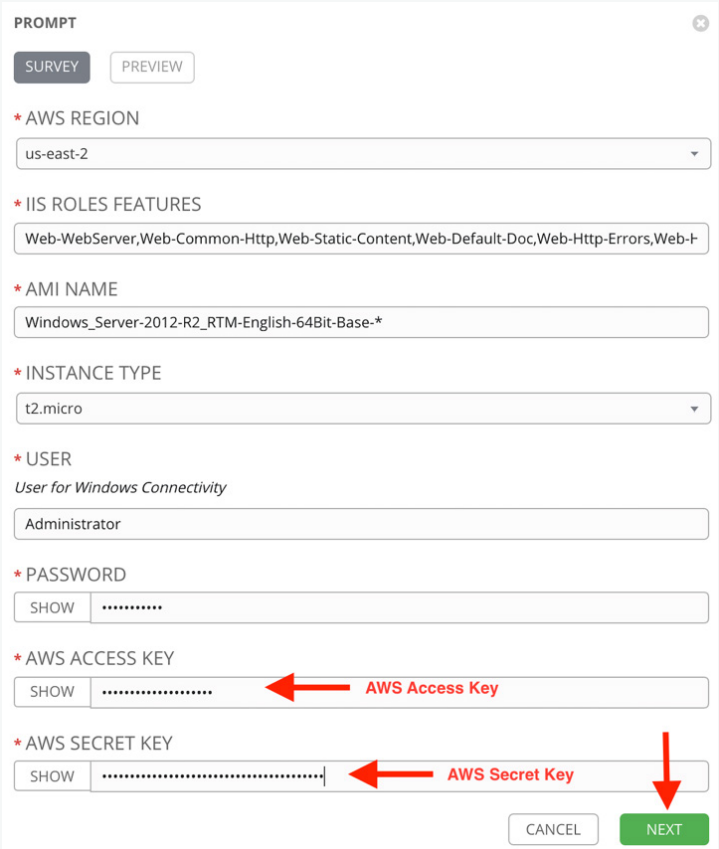


Executing Job from Template

1. After clicking ‘SAVE’, you should now see the template in your list of templates that you can execute to start a job.



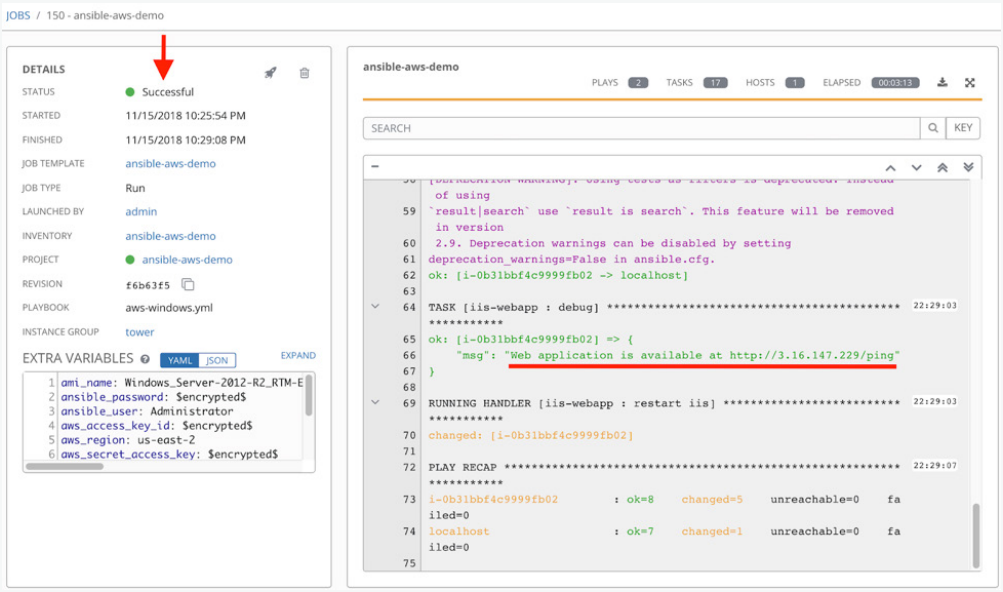
2. Click on the white and blue rocket to start the job. The survey prompt will come up immediately to gather the variables values.



3. Next you will proceed to the preview. Ansible Tower has taken the variable information you provided and put it into YAML format.

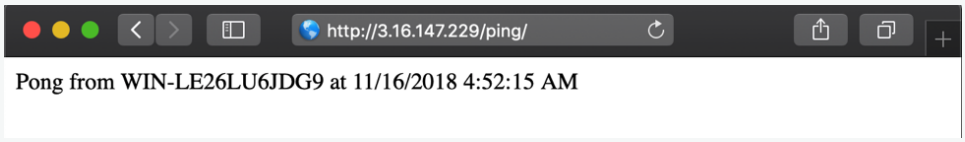
```
---
aws_region: us-east-2
iis_roles_features: >-

Web-WebServer,Web-Common-Http,Web-Static-Content,Web-Default-Doc,Web-Http-Er
rors,Web-Http-Redirect,Web-ASP,Web-Http-Logging,AS-Web-Support
ami_name: Windows_Server-2012-R2_RTM-English-64Bit-Base-*
instance_type: t2.micro
ansible_user: Administrator
ansible_password: $encrypted$
aws_access_key_id: $encrypted$
aws_secret_access_key: $encrypted$
```

4. Upon completing the review, click ‘NEXT’. The job will now begin to execute. Sometimes it takes a minute or two for the execution log to populate, and you may need to refresh the page.

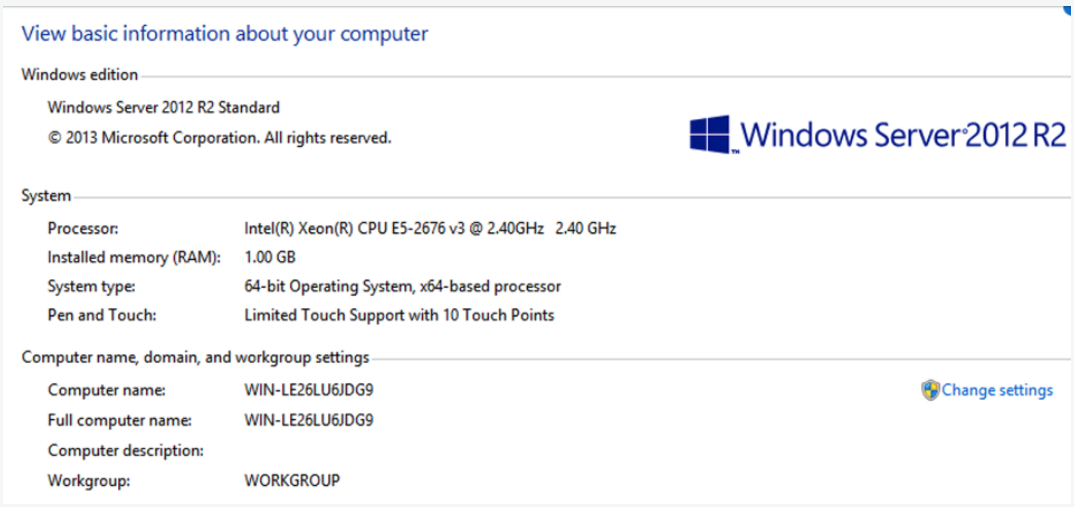


Deployment Verification

As you can see in the image above, the job completed successfully and also completed the test to ensure the application is online. This is all explained in the code review section. You can enter the url provided by the output of the execution log and insert in to a browser.



You can also verify that the server was deployed successfully by accessing the machine via RDP. The credentials are the ones specified in the survey prompts. As you can see, the response from the application is running in IIS, and the server name matches the hostname from the server properties I've displayed from the RDP session.



Before running the playbook:

Name	BelongsTo	Instance ID	Instance Type	Availability Zone	Instance State
AnsibleTower330	Daniel Foley	i-[redacted]	t2.medium	us-east-2a	running

After running the playbook:

Name	BelongsTo	Instance ID	Instance Type	Availability Zone	Instance State
AnsibleTower330	Daniel Foley	i-[redacted]	t2.medium	us-east-2a	running
ansible-managed		i-[redacted]	t2.micro	us-east-2c	running

Code Review

We're going to take a quick look at this demo playbooks code to see how it deploys an AWS EC2 Windows AMI, waits for it to become available, then installs IIS Features, deploys an application, and runs a test to ensure it's up and running.

aws-ansible-demo/aws-windows.yml

```
---

- name: "Launch Windows AMI."
  hosts: localhost

  roles:
  - aws-windows

- name: "Setup web application."
  hosts: windows

  roles:
  - iis-webapp
```

There are two roles in this playbook, the first one is called 'aws-windows', and it executes on the localhost, which is all that is required to query the AWS API.

AWS-WINDOWS Role

aws-ansible-demo/roles/aws-windows/tasks/main.yml

```
---

- name: "Find current, region-specific Windows AMI." - 1.1
  ec2_ami_find:
    aws_access_key: "{{ aws_access_key_id }}"
    aws_secret_key: "{{ aws_secret_access_key }}"
    region: "{{ aws_region }}"
    platform: windows
    virtualization_type: hvm
    owner: amazon
    name: "{{ ami_name }}"
    no_result_action: fail
    sort: name
    sort_order: descending
  register: found_amis
```

```
- set_fact:
    win_ami_id: "{{ (found_amis.results | first).ami_id  }}" - 1.2
  when: found_amis.results is defined

- name: "Create security group for WinRM and RDP." - 1.3
  ec2_group:
    aws_access_key: "{{ aws_access_key_id }}"
    aws_secret_key: "{{ aws_secret_access_key }}"
    name: "WinRM RDP"
    description: "Inbound WinRM and RDP"
    region: "{{ aws_region }}"
    rules:
    - proto: tcp
      from_port: 80
      to_port: 80
      cidr_ip: 0.0.0.0/0
    - proto: tcp
      from_port: 5986
      to_port: 5986
      cidr_ip: 0.0.0.0/0
    - proto: tcp
      from_port: 3389
      to_port: 3389
      cidr_ip: 0.0.0.0/0
  register: security_group

- name: "Start Windows instances." - 1.4
  ec2:
    aws_access_key: "{{ aws_access_key_id }}"
    aws_secret_key: "{{ aws_secret_access_key }}"
    region: "{{ aws_region }}"
    image: "{{ win_ami_id }}"
    instance_type: "{{ instance_type }}"
    group_id: "{{ security_group.group_id }}"
    wait: yes
    wait_timeout: 500
    exact_count: 1
    count_tag:
      Name: ansible-managed
    instance_tags:
      Name: ansible-managed
    user_data: "{{ lookup('template', '{{ role_path }}/templates/userdata.txt.j2')
}}"
  register: ec2_result
```

```
- name: "Wait for WinRM on all hosts." - 1.5
  wait_for:
    port: 5986
    host: "{{ item.public_ip }}"
    timeout: 300
  with_items: "{{ ec2_result.tagged_instances }}"

- name: "Add hosts to inventory." - 1.6
  add_host:
    name: "{{ item.id }}"
    ansible_host: "{{ item.public_ip }}"
    groups: windows
  changed_when: false
  with_items: "{{ ec2_result.tagged_instances }}"
```

The tasks in the ‘aws-windows’ role prepare AWS EC2, finds the AMI requested in the survey, starts the Windows instance while using the ‘userdata.txt.j2’ template to run a simple Powershell script that enables PSRemoting, and sets the Administrator password. It waits for the instance to come available, then dynamically adds the instance to the ‘windows’ inventory group we created.

- 1.1 Find the AMI in AWS, export the results to a variable
- 1.2 Parse the AMI output from step 1.1 in order to obtain the AMI ID
- 1.3 Create the AWS EC2 security group for WinRM and RDP connectivity
- 1.4 Deploy and start the AMI using the ID obtained in step 1.2, tag the instance and specify user_data which is a powershell script that sets the Administrator password and enables PSRemoting
- 1.5 Wait for WinRM on port 5986 to become available, wait up to 5 minutes before failing
- 1.6 Once WinRM is available, you will have confirmation the Windows AWS instance is up and running; now, dynamically add the host to the ‘windows’ group in the inventory we created earlier

IIS-WEBAPP Role

aws-ansible-demo/roles/iis-webapp/tasks/main.yml

```
---

- block:

  - name: "Install IIS, roles, and features." - 2.1
    win_feature:
      name: "{{ iis_roles_features }}"
      state: present
      restart: no
      include_sub_features: no
      include_management_tools: no
    notify: restart iis

  when: iis_roles_features is defined

- name: "Create ping directory." - 2.2
  win_file:
    path: c:\inetpub\ping
    state: directory

- name: "Create a default response page." - 2.3
  win_copy:
    src: "{{ role_path }}/files/default.aspx"
    dest: c:\inetpub\ping\default.aspx

- name: "Create ping web application." - 2.4
  win_iis_webapplication:
    name: ping
    physical_path: c:\inetpub\ping
    site: Default Web Site

- name: "Test ping web application." - 2.5
  uri:
    url: http://{{ ansible_host }}/ping
    return_content: yes
  register: uri_out
  delegate_to: localhost
  until: uri_out.content | search("Pong from")
  retries: 3

- debug: - 2.6
    msg: Web application is available at http://{{ ansible_host }}/ping
```

The ‘iis-webapp’ role is specifically for configuring the Windows server, deploying the simple IIS web application, and then verifying it has deployed successfully.

- 2.1 Install the Windows features when the variable is defined (if the variable is not defined, that step is skipped)
- 2.2 Create a directory for our application in the windows IIS default subdirectory
- 2.3 Copy our default home directory from the files in our Ansible Playbook, stored here: ‘aws-ansible-demo/roles/iis-webapp/files/default.aspx’
- 2.4 Create the actual web application in IIS
- 2.5 Test to make sure our application responds after 3 tries; wait for the response to include the words “Pong from”
- 2.6 Debug response. If all of the steps above completed successfully, show that the web application is available at what URL

LEVVEL

About Levvel

We are an IT, strategy, and design consulting firm that combines the innovative DNA of a startup with the wisdom, scalability, and process rigor of a Fortune 100 company.

Levvel's proven DevOps Assessment methodology delivers a precise strategy to enable your team to deliver on key practices that help organizations innovate faster through automating and streamlining the software development and infrastructure management processes.

We firmly believe that mentoring can be integrated with delivery. Our main focus is on saving our partners as much as possible on the lifetime total cost of ownership and maintainability of their systems. For more information, contact us at hello@levvel.io.