Article
Murray Oldfield · Feb 26, 2016 13m read

Provision a Caché application using Ansible - Part 1

Ansible helped me solve the problem of quickly deploying Caché and application components for Data Platforms benchmarks. You can use the same tools and methodology for standing up your test labs, training systems, development or other environments. If you deploy applications at customer sites you could automate much of the deployment and ensure that system, Caché and your application are configured to your applications best practice standards.

Overview

As a Technology Architect one of our groups responsibilities is to benchmark InterSystems Data Platforms on various vendors hardware and operating systems. Often the infrastructure is pre-release and we have limited time before it must be returned or handed over to someone else so its vital to get the benchmark set up quickly and accurately so we get as much time as possible to do the real benchmark work.

Over the years we automated a lot of the benchmark install tasks with shell script-lets and cut and paste from cheat sheets and check lists, but thats very intensive and prone to errors, especially when you have many servers and bounce around between different operating systems - the differences between installing or using services on SLES 11, Red Hat 6, Red Hat 7, AIX etc can be subtle and annoying.

I chose Ansible for the task of provisioning Data Platforms applications and the benchmark components after looking at several software options available for automating configuring and managing systems. It is important to note that I am not mandating that Ansible is **THE** solution for deployment and configuration. I looked at the features and operation of other tools such as Puppet and Chef before I chose Ansible. If your organisation is already using one of the other tools you can use them — the methodology, commands etc I use with Ansible should be translatable to the other software, my hope is that these posts will help you no matter what tools you are using.

This is the first post in a series that will show using Ansible when deploying InterSystems Data Platforms applications. This post covers laying the foundation by installing Caché, the next post will extend the solution to include application installation including using the %installer class. This post covers:

- Overview and installation of Ansible
- Layout of Ansible for easy administration and scaling.
- Install Caché on one or more servers concurrently.

What is Ansible?

Ansible allows you to configure one or more servers while automating complex tasks and allows you to add new servers very simply. Tasks are designed to be idempotent (you can run the same scripts any number of times on the same server and the resulting server config will be the same).

A key reason I selected Ansible for the provisioning task was the minimal system requirements (Python 2.7 which is on Linux servers already) and that it is a self contained solution — Ansible code is only installed on a control server and uses a a push architecture to run commands and scripts on target servers using OpenSSH. There are no agents required on the servers being provisioned. As a counterpoint Chef and Puppet are a pull architecture where software is loaded on client servers (web, DB, etc) with clients continuously polling the master for updates. Ansible's push architecture also lends itself to a progressive implementation of servers as your schedule demands.

Ansible is open source, and maintained by the community. Ansible, Inc has been owned by Red Hat since 2015. Ansible, Inc has a premium lifecycle product (Ansible Tower) and support and training that they charge for, however everything in the post uses the open source command line version. There is an active community (Ansible Galaxy) with many pre-made solutions you can download for a large number of tasks such as installing web servers, ftp, kerbros, the list goes on. For example for the full benchmark deployment project I include an Apache module I downloaded and customised to install and configure Apache 2.x on RHEL, SLES or Solaris (along with other platforms).

Instructions for downloading and installing Ansible can be found on the Ansible web site and github. There is an active community if you have question or wish to contribute.

https://www.ansible.com/get-started
http://docs.ansible.com

Installing Ansible

The examples in this post have been tested on VMs running Red Hat 7.0 and 7.2 - I also did initial testing on my laptop with Centos 7 for the Ansible controller server using virtual box and vagrant. Caché is not required to be installed on the controller so your choice of operating system is more than Caché supported platforms list. To keep things simple I used the current rpm version of Ansible available for Red Hat (Ansible 1.9.4), later versions are available from github.

In the example I am installing cache-2015.2.2.805.0-lnxrhx64 - but the same general procedures apply to HealthShare or Ensemble distributions. As you will see later we parametrise install options with variables for specific file names, directory paths, and so on.

In this first post I have cut the tasks down to a basic Caché install so most of the tasks are platform independent. When an Ansible playbook starts one of the first tasks is to get a manifest of the target machine — operating systems, interface cards, memory details, number of CPUS, disk layout, and so on, this knowledge of the target operating system is used when commands are run to abstract the commands in Ansible scripts from the actual command run on the target (e.g. service start httpd on Red Hat vs /etc/init.d/apache2 restart on SLES).

I will assume that you have read the instructions and installed Ansible on your control machine as per the instructions for your platform.

http://docs.ansible.com/ansible/introinstallation.html

Ansible must use a Linux system as the controller, but the target systems can be Linux or Windows. More information about having a windows target is available from the Ansible documentation.

http://docs.ansible.com/ansible/introwindows.html

Example controller system install: Ansible on RHEL/CentOS 7 64 Bit

On Red Hat or CentOS you must first install the epel-release (Extra Packages for Enterprise Linux) RPM which includes Ansible. The epel project is designed for major Linux distributions by providing a bundle of many useful open source packages (networking, system administration, monitoring, etc.).

```
[root@localhost tmp]# wget http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-5.noarch.rpm
:
```

```
:
[root@localhost tmp]# rpm -ivh epel-release-7-5.noarch.rpm
:
:
[root@localhost tmp]# yum --enablerepo=epel info ansible
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-manager
Installed Packages
Name : ansible
Arch : noarch
Version : 1.9.4
Release : 1.el7
Size : 7.0 M
Repo : installed
From repo : epel
Summary : SSH-based configuration management, deployment, and task execution system
URL : http://ansible.com
License : GPLv3+
Description :
 : Ansible is a radically simple model-driven configuration management,
 : multi-node deployment, and remote task execution system. Ansible works
 : over SSH and does not require any software or daemons to be installed
 : on remote nodes. Extension modules can be written in any language and
 : are transferred to managed machines automatically.
[root@localhost tmp]#
[root@localhost tmp]# sudo yum install ansible
:
:
[root@localhost tmp]# ansible --version
  ansible 1.9.4
  configured module search path = None
```

Cool... Readty to start!

Ansible Orientation

The Ansible documentation should be reviewed for the use of the different Ansible components; Inventory, Playbook, Modules, Roles etc.

To make administration simpler and to avoid large and complex script files a predefined directory structure and search paths are used. For this post I am going to use a file structure that uses Ansible recommendations and can be used as a model when you think about building out the example for bigger installations.

The ansible modules used are commented and self explanatory and are available on github. Download the files and read through to get a feel for the workflow.

https://github.com/intersystems/ansible-deploy-cache

The base directory for our example has the following files;

- ansible.cfg: Changes from the ansible defaults.
- inventory: Defines and describes the working environment. e.g. server names/IPs.
- <various>.yml files: These files describe the task sets that will be run for a particular server role.

Terminology

To make later discussions clearer a quick explanation of some of the Ansible terminology follows.

Modules are the building blocks that make automation actions you perform on systems. Each module is built for specific tasks and parameters can be used to change that task. For example copy files, create user, run commands, start services, etc etc. Currently there are more than 400 modules included in the default Ansible install, plus many more from community, or you can create your own.

Modules are combined together to make plays and playbooks as a way to execute automation workflow. A play can have multiple tasks and a playbook can have multiple plays.

Roles allow you to combine Playbooks. A role can be thought of as grouping together configuration of components for a server depending on the target servers use. In the examples in this post roles build up layers of configuration to build a server.

In my benchmarking set up I have the following roles to build servers with:

- hsservercommon: Configure OS, install Apache, install Caché.
- webserver: Copy web files (csp, html, js, etc), configure Apache for application.
- generator: Copy files, create and configure webstress generator databases, namespaces, global mapping, etc.
- dbserver: Copy files, configure DB server system settings, application databases, namespaces, global mapping, etc.

These roles can be combined to build different server types:

- hsservercommon + webserver + generator = a webstress generator server.
- hsservercommon + webserver = an application web server.
- hsservercommon + dbsevrer = a database server.

What makes up a role and what configuration is included in each role will be very specific to the application being deployed. The example in this post will use a minimal set of tasks which assume pre-configuration of the operating system, however with the modules available on Ansible and Galaxy much more complex and full featured system configuration is possible.

Notes on installing Caché

I have written the examples to introduce some interesting and useful features, the following are some highlights. Note: The examples can be used as a guide to install InterSystems Data Platforms: Caché, HealthShare and Ensemble. I wrote the examples to install HealthShare, in the examples HealthShare and Caché have the same actions.

./testserver/roles/hsservercommon/tasks/main.yml

This is the mainline script for the common tasks of Configure OS, install Apache, install Caché. For this post I have trimmed back to just the include files for Red Hat and to install and configure Caché. You can see that after starting Ansible has knowledge of the operating system kept in *ansible_** variables, including *ansible_os_family* which we can use to make decisions in our script.

./testserver/roles/hsservercommon/tasks/configure-healthshare2015.yml

This is the main script for installing Caché. Looking through the script you see a logical workflow for tasks on the target including;

- Create operating system users and groups.
- Copy install files from manifest folder on the controller.
- Uncompress install files.
- install Caché using silent install (see note below).
- Copy Caché key file
- Set default Caché instance
- Restart Apache
- Restart Caché

There are several options for a silent install of Caché, including;

- Using a parameters.is file. A template .isc file is created by a previous install and can be used as-is or modified.
- Using cinstallsilent with key-value pairs set in environment.
- Using %installer class.

For this example I chose to use installsilent however I have also included a commented out alternative using a parameters file to show how templates files can be used in Ansible (see ./roles/hsservercommon/templates/parametershs20152rh64.isc).

In later posts I will show using the %installer class for installing Caché and for setting up database and namespaces. Details on the install options are available in the Caché online documentation, there is also an excellent post on the community for using the %installer class.

A parameters file is useful when you want to install and configure Caché for use CSPGateway with a web server other than the Caché internal apache version in older versions of Caché. This functionality is available in %installer from Caché 2016.1.

./testserver/roles/hsservercommon/tasks/setupRedHat.yml

This example is included to show use of system specific variables (ansible*) and setting operating system variables.

./testserver/roles/hsservercommon/vars/*

Variables files contain variable in key: value pairs and as you can see are a way to reuse the same scripts for different environments and situations.

Running Caché install

For this example I will assume the systems are available and set up as follows.

1. The controller has Ansible installed and the following directories populated with files and structure from github.

- **./testserver/*** : directory tree with inventory, .yml files etc. Including...
- **./testserver/DistributionFiles/Cache** : (manifest containing Caché distribution and cache.key).

2. Target machines have Red Hat installed and Apache installed.

You will need to edit the following files to customise the install for your test environment.

1. **inventorytest**

You will need to edit for your test server names or ip addresses.

2. **./testserver/roles/hsservercommon/vars/healthshare2015.yml**

You must edit the paths to fit your test environment. Look at the following paths for the target servers:

- **commoninstallbasepath** : where manifest files are copied to, unpacked, and Caché install runs.
- **ISCPACKAGEINSTALLDIR** : Caché install directory

The directory paths will be created on the target server if they don't already exist.
NOTE: One of the features of an automated deployment is to build multiple servers in parallel. If there are multiple servers in your inventory file each step runs to completion concurrently on each target server before the next step starts on each of the servers in the group. If a step fails on any server the script stops. You will see an error message to help you correct the problem. Once the error is corrected you then simply rerun from the beginning — this is a key feature of the scripts — scripts are designed to be idempotent. Idempotent means that when a module is run in a step, for example copy a file, if the file exists already the step is not rerun, the script just moves on to the next step. Modules like copy do have parameters you can set to force a copy, but that is not the default. Closer examination of he scripts will show that the "creates" argument is used in some cases, for example:

```
- name: unattended install of hs using cinstall_silent
 shell: >
 ISC_PACKAGE_INSTANCENAME="{{ ISC_PACKAGE_INSTANCENAME }}"
 ISC_PACKAGE_INSTALLDIR="{{ ISC_PACKAGE_INSTALLDIR }}"
 ISC_PACKAGE_UNICODE="{{ ISC_PACKAGE_UNICODE }}"
 ISC_PACKAGE_INITIAL_SECURITY="{{ ISC_PACKAGE_INITIAL_SECURI
TY }}"
 ISC_PACKAGE_MGRUSER="{{ ISC_PACKAGE_MGRUSER }}"
 ISC_PACKAGE_MGRGROUP="{{ ISC_PACKAGE_MGRGROUP }}"
 ISC_PACKAGE_USER_PASSWORD="{{ ISC_PACKAGE_USER_PASSWORD }}"

 ISC_PACKAGE_CACHEUSER="{{ ISC_PACKAGE_CACHEUSER }}"
 ISC_PACKAGE_CACHEGROUP="{{ ISC_PACKAGE_CACHEGROUP }}" ./cin
stall_silent
 chdir="{{ common_install_base_path }}/{{ hs_install_unpack_
path }}"
 args:
 creates: "{{ ISC_PACKAGE_INSTALLDIR }}/cinstall.log"
```

The Stanza above uses the creates argument to tell the Ansible module (shell module in this case) that this action creates the cinstall.log file. If the module finds this file (Caché is already installed) this step will not be run.

OK, once your all set we can run the install.

```
$ ansible-playbook dbserver.yml
PLAY [dbservers] ***************************************
```

```
********************
GATHERING FACTS ***************************************************
********************
ok: [db1]
TASK: [hs_server_common | include_vars healthshare2015.yml]
********************
ok: [db1]
TASK: [hs_server_common | include_vars os-
RedHat.yml] ************************
ok: [db1]
etc
etc
etc
TASK: [hs_server_common | Create default cache group] ******
********************
changed: [db1]
TASK: [hs_server_common | Create default cache manager group
] ******************
changed: [db1]
TASK: [hs_server_common | Create default cache user] *******
********************
changed: [db1]
TASK: [hs_server_common | Create default cache system users]
 ********************
changed: [db1]
TASK: [hs_server_common | Create full hs install temp direct
ory] **************
changed: [db1]
TASK: [hs_server_common | Check tar file (gunzipped already)
 does not exist] ***
ok: [db1]
TASK: [hs_server_common | Copy healthshare install file] ***
********************
changed: [db1]
TASK: [hs_server_common | un zip hs folder] ****************
********************
changed: [db1]
TASK: [hs_server_common | un tar hs install] ***************
********************
changed: [db1]
TASK: [hs_server_common | Create hs install directory] *****
********************
```

```
changed: [db1]
TASK: [hs_server_common | touch ztrak.conf.] ***************
********************
changed: [db1]
TASK: [hs_server_common | Process parameters file] *********
********************
changed: [db1]
TASK: [hs_server_common | unattended install of hs using cin
stall_silent] *****
changed: [db1]
TASK: [hs_server_common | copy hs key] **********************
********************
changed: [db1]
TASK: [hs_server_common | Set default hs instance] *********
********************
changed: [db1]
TASK: [hs_server_common | restart apache to initialize CSP.i
ni file] **********
changed: [db1]
NOTIFIED: [hs_server_common | restart healthshare] *********
********************
changed: [db1]
PLAY RECAP ************************************************
********************
db1 : ok=32 changed=21 unreachable=0 failed=0
```

If we look on the target server — the db server Caché is now up and running.

```
$ ccontrol list


Configuration 'H2015' (default)
 directory: /test/hs2015
 versionid: 2015.2.1.705.0
 conf file: cache.cpf (SuperServer port = 1972, WebServer =
57772)
 status: running, since Wed Feb 17 15:59:11 2016
 state: ok
```

Summary

In subsequent posts I will build out the scripts with other tasks such as editing configuration files and using the %installer class to configure an application.

If you find this interesting and start to create your own deployments please feel free to contact me with questions or suggestions. I am a regular speaker at Global Summit on virtualisation and performance - so if you are attending Global Summit this year please introduce yourself, I am more than happy to chat about your experiences with Ansible or any other system architecture topics.

#Best Practices #Open Source #System Administration #Caché