An Intro to Configuration Management

DevOps 102

What is Configuration Management?

- Configuration Management refers to the process of systematically handling changes to a system in a way that maintains integrity over time.
- Automation plays an essential role in configuration management. It is the mechanism by which the server reaches its (declarative) desired state, which was previously defined by using (imperative) provisioning scripts in a tool-specific language.
- The CM process is widely used by military engineering to manage changes throughout the lifecycle of complex systems, such as weapon systems, military vehicles, and information systems. Outside the military, the CM process is also used with IT service management, civil engineering and other industrial engineering segments such as roads, bridges, canals, dams, and buildings.

Why do we need Configuration Management?

- CM applied over the lifecycle of a system provides visibility and control into its performance, functional, and physical attributes. CM verifies that a system performs as intended, is configured and documented in sufficient detail to support its projected life cycle.
- The relatively minimal cost of implementing CM is returned many fold in cost avoidance. The lack of CM, or its ineffectual implementation, can be very expensive and sometimes can have such catastrophic consequences such as failure of equipment or loss of life.
- CM emphasizes the functional relationship between parts, systems, and subsystems for effectively controlling system change. Changes to the system are proposed, evaluated, and implemented using a standardized, systematic approach that ensures consistency, and proposed changes are evaluated in terms of their anticipated impact on the entire system (i.e. "a diff"). CM verifies that changes are carried out as prescribed and that documentation of items and systems reflects their true configuration.
- A complete CM strategy ensures that documentation (e.g., requirements, design, test, and acceptance documentation) for items is accurate and consistent with the actual physical design of the item. In many cases, without CM, the documentation exists but is not consistent with the item itself. For this reason, employees are frequently forced to develop documentation reflecting the actual status of the environment before they can proceed with a change. This reverse engineering process is wasteful in terms of human resources and can be minimized or eliminated by using CM.

Misconceptions

What it is

- An automation tool
- Mutable (can be changed)
- For human tasks
- A traditional approach to network automation and change management
- Pets

What it's not

- A pipeline or a deployment tool
- Immutable (unchanging)
- For machine tasks
- A modern DevOps approach to change management
- Cattle

About This Course

Goals

- Introduce Git, and familiarize oneself with core concepts
- Introduce Bash scripting, and write a simple script
- Familiarize oneself with the VSCode IDE, task management, and extensions
- Deep-dive into Saltstack concepts and usage

Prerequisites

- GitKraken
- VSCode
- Bitbucket

Connect to the Git Server

Link GitKraken to Bitbucket

1. Open GitKraken. Navigate to File > Preferences > Authentication.
2. Click on "Bitbucket Server". Input the following information:
    1. Host Domain: http://bitbucket.dal.$REDACTED.net/
3. Click on "Generate a token on the Bitbucket Server".
4. Once redirected, give your token a name (such as "Ryan-WorkPC"). Grant the following permissions:
    1. Projects: Admin (or Write)
    2. Repositories: Admin (or Write)
5. Click on "Create." Copy your new token and paste it into GitKraken. Then click on "Connect".
6. Click on "Generate SSH key", then "Manage SSH keys on Bitbucket server". Add your new public key on the server.

Clone the "Salt" repository

1. Click on File > Clone Repo > Bitbucket Server.
2. Click "Browse" to choose a local directory to copy your repo to.
    1. I would recommend C:\Users\<youruser>\Documents\Repos
3. Under the "STOP" project, choose "Salt". Click "Clone the repo!"

Checkout a Personal Branch

Check out the “development” branch

1. In GitKraken, double-click the “development” branch under “Remotes”.
2. You now have a “local” copy of the latest code from the “development” branch.

Create a personal branch

1. Within GitKraken, click on “Branch”. Give your branch a unique name, such as “Feature-XYZ” or “Ryan-$REDACTED”.
2. You are now ready to work with this code.

Make Your First Change

Connect VSCode to your project

- Open up VSCode, and navigate to File > Open Folder.
- Choose your local "Salt" repository, then click on "Open."
- You're now ready to work with this repo.

Make a change

- Open up "README.md."
- Make a trivial modification – then press "Ctrl+S" to save the file.
- Return to GitKraken. You will notice that GitKraken is aware of your changes – and will show you a "diff" of the current and new file.
- To save your changes:
  - "Stage" the modified file. (i.e. "git add")
  - Add a brief commit message, then commit (i.e. "git commit")
  - Push your changes to the Remote server (i.e. "git push")

Merge Your Commits into Development

Open a Merge Request/Pull Request

- In GitKraken, click on the "+" next to "Pull Requests".
- Choose "Bitbucket server." Use the following settings:
  - From Repo: <STOP/Salt>
  - Branch: <yourbranch>
  - To Repo: <STOP/Salt>
  - Branch: development
  - Pull Request Title: <a brief description of your changes>
  - Pull request description: <optional but recommended>
- Click on "Create pull request".
- Now, you must wait for administrator approval to "merge" your changes into the "development" branch.
- Repeat indefinitely.

Write Your First Shell Script

Overview

- Bash is among the most simple of all scripting languages. At its core, Bash has three main components:
  - Commands
  - Variables
  - Logic
- We will be skipping logic in this tutorial.

Definitions

- **Commands**: if it can be run in Linux, it can be run in Bash. If you have ever executed a command on a Linux server – that command may be dropped into a Bash script with no further action required.
- **Variables**: "Changing" or "variable" pieces of information.
  - String:
    - $HOSTNAME = c2ssvc01
  - Boolean:
    - $IS_ACTIVE = True/False (1/0)
  - Integer:
    - $APPLE_COUNT = 25
  - List:
    - $FRUITS = "Apple, Orange, Banana"

Write Your First Shell Script

Steps

1. Create a file named “test.sh”.
2. Add the shebang line. This tells Bash where to look for the interpreter.
3. Add your name to the variable.
4. Add the command you would like to execute.
5. Save the script, and execute:
    1. bash test.sh

Example

test.sh

```
1  #!/bin/bash
2
3  MY_NAME="Ryan"
4
5  echo "Greetings, $MY_NAME!"
```

A Note About Shell Scripts

- When "logic" is used, scripts can become complicated and hard to understand.
  - Example:
    - ./salt/oracle/clone/files/setup/CLONE_SETUP_07012019.sh
    - ./salt/rapid/menu.sh
- However, when scripts are written in a declarative fashion, they are simple, single-purpose, and easy to understand.
  - Example:
    - ./salt/oracle/clone/files/clone-setup.sh
    - ./tasks/vscode/salt-master-config.ps1
- When declarative scripts are coupled with Configuration Management – they can be scaled to infinity:
  - Example:
    - ./pillar/oracle/clone.sls

Salt: Two Modes of Operation

Masterless Mode

- In this mode, minions are configured directly. There is no master handing-out changes.
- This mode is push-based; configurations are pushed to servers directly, usually via pipelines and continuous deployment tools.
- Orchestrations are difficult, because servers are not part of a shared system; servers are not aware of each other. Orchestrations are typically performed by other tools.

Masterful Mode

- In this mode, minions are controlled and configured by a master server. Minions are configured to sit idle until they are given instructions.
- This mode is pull-based. Minions "reach-out" to the master for instructions.
- Orchestrations are fully-supported within the Salt ecosystem, but it may be difficult to introduce other tooling, such as pipelines and continuous deployment. Other tooling is complicated by the interconnected nature of masterful mode, authentication requirements, multi-server dependencies, etc.

Salt: Configuration Files

Configuring minions

- Minions are primarily configured via files in the following locations:
  - /etc/salt/minion
  - /etc/salt/minion.d
- However, STOP minions are managed exclusively by the Linux Team, via files located at:
  - /opt/salt/sms/etc/salt/minion
  - /opt/salt/sms/etc/salt/minion.d
- To restart a minion, run:
  - service salt-sms-minion restart

Configuring masters

- Masters are primarily configured via files in the following locations:
  - /etc/salt/master
  - /etc/salt/master.d
- The Linux Team is responsible for a few minor settings. Otherwise, management is left to STOP. Their changes may be found in:
  - /opt/salt/sms/etc/salt/master
  - /opt/salt/sms/etc/salt/master.d
- To restart the master, run:
  - pkill salt-master
  - salt-master -d

Salt: Updating the Master Configuration

The Takeaway

- Essentially, you should only ever need to update configuration on the master.
- The configuration file we use is located in the "Salt" repository, under:
  - ./etc/salt/master.d/master.conf
- The default minion and master configuration files for any given version of Salt may be found at:
  - https://docs.saltstack.com/en/latest/ref/configuration/examples.html

Updating the master config

- The simplest way to update the Salt master is via an automated script:
  - This script will SSH into the Salt master, replace the configuration file with the one on your machine, and restart the Salt master daemon.
- This script may be executed in two ways:
  - Run the Powershell code directly via:
    - ./tasks/vscode/salt-master-config.ps1
  - Run the VSCode Task via Ctrl+Shift+P:
    - .vscode/tasks.json
    - Tasks: Run Task
    - Salt: Replace master configuration file

Salt: Running Arbitrary Commands

Overview

- Salt may be used to execute any command across one or more servers. To do so, one must run the commands from the master server.
- Example:
  - salt ‘d2gw21’ test.ping
  - salt ‘d2gw21’ cmd.run ‘df -h’
- Multiple servers may be managed via targeting. In these examples, we use “globbing”:
  - salt ‘d2*’ test.ping
  - salt ‘d2gw*’ cmd.run ‘du –sh’

Using VSCode Tasks

- Similar to the process for updating the master configuration file, these arbitrary commands may be executed via a VSCode task and Powershell script:
  - Run the Powershell code directly via:
    - ./tasks/vscode/salt-master-state.ps1
  - Run the VSCode Task via Ctrl+Shift+P:
    - .vscode/tasks.json
    - Tasks: Run Task
    - Salt: Execute a state

Salt: States

Overview

- Even though an administrator may run commands directly against minions, Salt really wants to be used in a declarative fashion (i.e. configuration data is written explicitly in version control, and underlying logic "trues up" the environment.) This is where states come in.
- The core of the Salt State system is the SLS, or **S**a**L**t **S**tate file. The SLS is a representation of the state in which a system should be in, and is set up to contain this data in a simple format. This is often called configuration management.
- Salt states are located in the "Salt" repo, under ./salt.

The Top File

- Most infrastructures are made up of groups of machines, each machine in the group performing a role similar to others. To effectively manage these groups, an administrator must be able to create group-specific roles. For example, a group of machines serving as a gateway might have roles indicating that those machines gateway service should always be running.
- In Salt, the file which contains a mapping between groups of machines on a network and the configuration roles that should be applied to them is called a top file.
- Top files are named top.sls by default and they are so-named because they always exist in the "top" of a directory hierarchy that contains state files. That directory hierarchy is called a state tree.
- Top files have three components:
  - **Environment:** A state tree directory containing a set of state files to configure systems.
  - **Target:** A grouping of machines which will have a set of states applied to them.
  - **State files:** A list of state files to apply to a target. Each state file describes one or more states to be configured and enforced on the targeted machines.

Salt: Pillars

Overview

- Salt pillar is a system that lets you define secure data that are 'assigned' to one or more minions using targets. Salt pillar data stores values such as ports, file paths, configuration parameters, and passwords.
- Succinctly, pillars are the data that is "fed into" Salt states. By comparison, pillars are the "variable" information that is passed to the declarative "states" or "scripts."
- Salt pillars are located in the "Salt" repo, under ./pillar.

The Top File

- Salt pillar uses a Top file to match Salt pillar data to Salt minions. This Top file is identical to the Top file that is used to match Salt states to Salt minions.

Salt: Highstate

Overview

- The Salt "state.highstate" is a special kind of Salt state that executes all states within all top files, across the entire infrastructure. 9 times out of 10, this is the command you will want to execute.
- Highstate is a two-step process:
  1. Generate declarative configuration files from the Salt states defined in top.sls. By default, states use the Jinja2 templating language.
  2. Apply the generated files to the infrastructure.

The Command

- The command to execute a highstate looks like this:
  - salt 'd2gw21' state.highstate
  - salt 'd*' state.highstate test=true
- Alternatively, this command is available as a VSCode Task:
  - Salt: Execute a state
- "state.apply" can be used to apply a single state. Used without any parameters, it behaves exactly like "state.highstate."

Salt: Orchestrations

Overview

- Executing states or highstate on a minion is perfect when you want to ensure that a minion is configured and running the way you want. Sometimes, however, you want to configure a set of minions all at once.
- For example, if you want to take a database clone first, then restart several webservers after, orchestrations will ensure that matching configuration is applied consistently across the entire service.

The Command

- The command to execute an orchestration looks like this:
  - salt-run state.orch orchestrate.oracle.clone saltenv=development --async

Salt: Demonstration

- Demonstrate the relationship between the Oracle clone Salt states, pillar data, and orchestration state.
- Show the original script, and show the re-designed declarative script.

The Emergent Properties of Declaration

- By writing scripts in a declarative fashion, we can think about problems as data points – not complicated logic that must be interpreted.
- Declarative scripting brings to light issues that may otherwise be obscured by too much logic.
- Configuration management tools are designed to be the pinnacle of well-designed, well-written code in the tool's native language (Salt is written in Python). The tool abstracts the details of the underlying code, providing a simple, modular management interface that enforces best practice code development.

Known Deficiencies

- Current development practices require that we commit known, broken code to git – just to test it.
- Salt cannot currently be tested in isolation. All changes must be tested in "standing" development environments. Anything that breaks must be troubleshooted – rather than disposed of and rebuilt.
- Changes may not be tested in development until after they have been merged into the "development" branch. This is cumbersome and will lead to frustration.
- Salt has system-level dependencies, which can vary from server to server (depending on that server's workload.) This can lead to inconsistencies in Salt behavior.
- Masters and Minions are sometimes not running the same version.

Questions?


WE TOOK THE HOSTAGES, SECURED THE BUILDING, AND CUT THE COMMUNICATION LINES LIKE YOU SAID.
EXCELLENT.
BUT THEN THIS GUY CLIMBED UP THE VENTILATION DUCTS AND WALKED ACROSS BROKEN GLASS, KILLING ANYONE WE SENT TO STOP HIM.
AND HE RESCUED THE HOSTAGES?
NO, HE IGNORED THEM. HE JUST RECONNECTED THE CABLES WE CUT, MUTTERING SOMETHING ABOUT "UPTIME".
SHIT, WE'RE DEALING WITH A SYSADMIN.