Progress Chef

Using Chef InSpec to Achieve Compliance Automation with Ansible

USER GUIDE

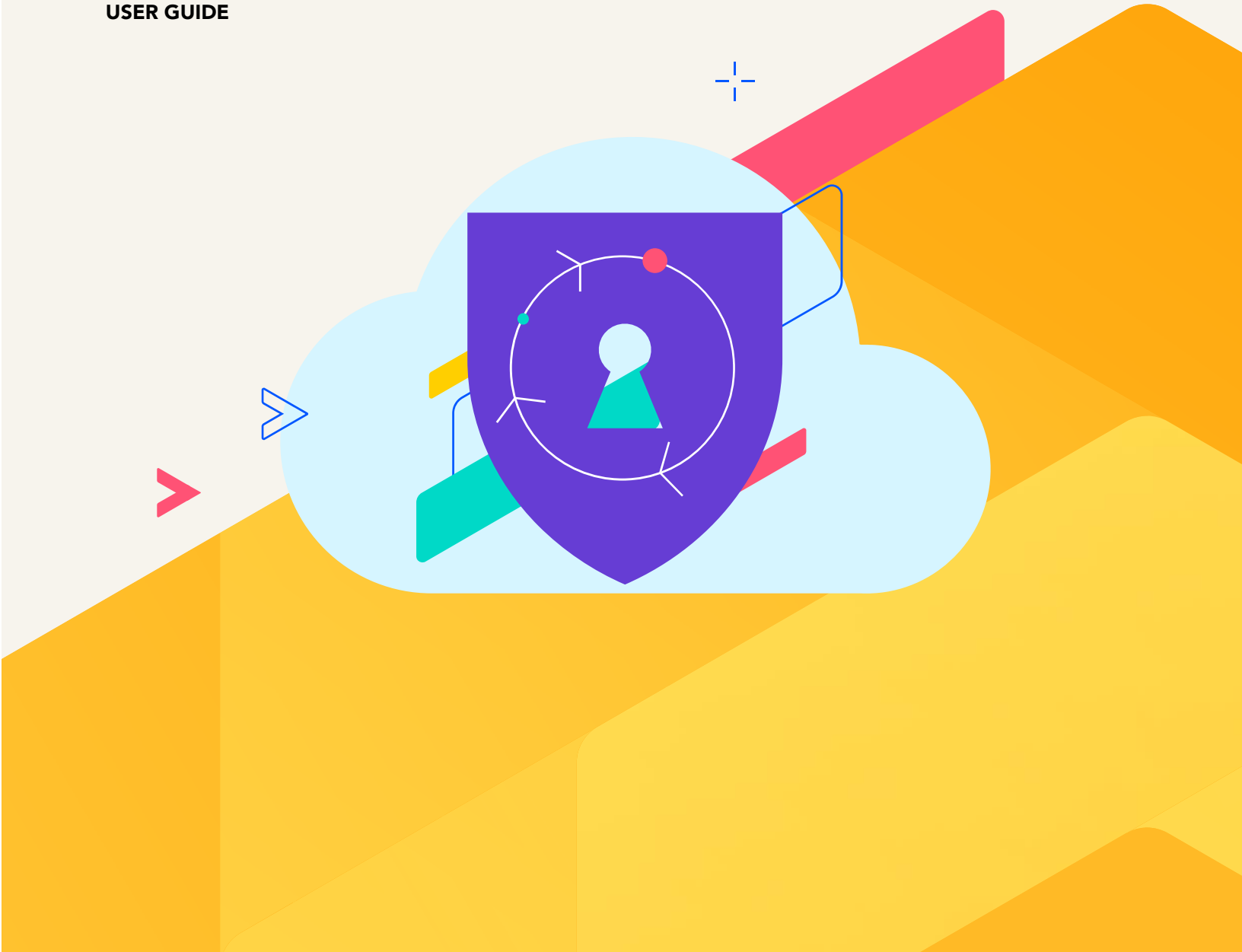


Table of Contents

`

Shift-Left with Compliance Automation

The concept of "shifting testing left" has grown in popularity, but many organizations often don't thoroughly test new code until it's being readied for production. Discovering issues at that late stage can delay releases, lead to unplanned work, and even cause costly rollbacks. The earlier code is tested in a DevOps workflow, the earlier problems can be addressed, saving time and reducing stress.

But testing is only part of the picture. Shifting security and compliance left can reap many benefits, often in use-cases where the stakes are high. To achieve continuous compliance, your organization must be able to detect misconfigurations and security flaws consistently throughout every stage of the software development cycle. This paper looks at three distinct kinds of detection:

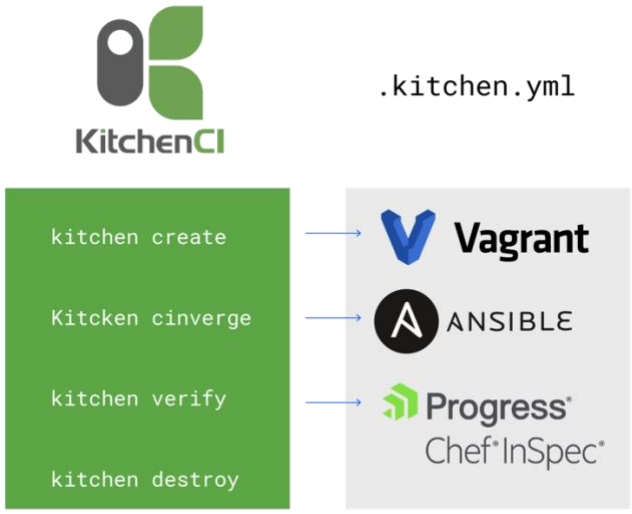
- **Code Testing:** Does my code behave the way I expect it to?
- **Security Assessment**: Is my environment vulnerable to known exploits?
- **Compliance Auditing**: Does my environment comply with defined compliance frameworks?

While there is some overlap between these concerns, they are distinct in their purpose and scope. Yet, they share one important quality: All three can be measured by assessing a running environment against a defined expectation. And of course, organizations benefit by assessing all three early in a product development cycle.

Chef InSpec is a tool designed to tackle all three areas and is unique in the industry as the only testing and compliance tool built from the ground up to be used by stakeholders across your organization, allowing security, operations, QA, and development to have a unified way of collaborating on requirements. Since InSpec only evaluates system state – and does not implement configuration changes – it can be run continuously as a means to automate your evaluation in every environment you manage. The following sections highlight how you can use InSpec alongside Ansible to ensure continuous compliance across your estate.

Code Testing: Validate Ansible Playbooks with InSpec and Test Kitchen

Test Kitchen is a testing system included in Chef Workstation (formerly ChefDK) to allow users to quickly evaluate configuration management code on ephemeral test infrastructure – like VMs, Docker containers and cloud instances – and then validate the results. It formalizes the testing process into four steps:



Within Test Kitchen, these steps can be run individually, or all at once via a single command: "`kitchen test`".

What's particularly unique about Test Kitchen is that you can customize each step via a variety of plugins, making it valuable even to organizations that aren't using Chef to automate infrastructure. The following example illustrates this by showing how Test Kitchen can be used to verify an Ansible playbook using Chef InSpec.

Test Webserver Automation with InSpec & Ansible

A common automation task is configuring a secure webserver. As mentioned, a good place to start is to define an expectation that can be used to measure success, for example, confirming that your webserver is listening on the correct port (443 and not the default 80), and serving valid content. With InSpec, these requirements can be defined as "resources," and within those resources, your expected results are defined by "matchers." The below content is saved in a local file in `tests/website-https_verify.rb`. For easy reference, all the code examples used in this white paper are available at https://github.com/chef/chef-examples/tree/main/chef-and-ansible.

```
describe port(443) do
  it { should be_listening }
end

describe http('https://localhost/', ssl_verify: false)do
  its('status') { should cmp 200 }
  its('body') { should match /Hello, world!/ }
end

describe ssl(port: 443).protocols('ssl3') do
  it { should_not be_enabled }
end

describe ssl(port: 443).protocols('tls1.2') do
  it { should be_enabled }
end
```

Note that these are simple functional tests that don't specify a particular implementation of your webserver. Whatever technology is in use, these resources will determine:

- Is my server listening on port 443?
- Does making a web request to *localhost* return a valid status?
- Does that web request return the expected content ("Hello, world!")?

This is important because it means you can validate systems regardless of how they were initially configured. Whether your systems are configured with a tool like Chef Infra or Ansible, manually configured as legacy or brownfield environments, with VM or AMI templates, or a combination, Chef InSpec can evaluate the running state of those systems consistently. This allows you to easily move from manual operations to automation while ensuring that all configurations are correct.

In Ansible, configurations are defined in YAML files, and an example playbook (named `website_https.yml`) to configure Apache and create certificates might look like this:

```
- hosts: myhost
  remote_user: root
  become: yes
  vars:
      conftext: |
        <VirtualHost *:443>
         DocumentRoot "/var/www/helloworld"
         SSLEngine on
         SSLCertificateFile /etc/apache2/certs/apache.crt
         SSLCertificateKeyFile /etc/apache2/certs/apache.key
          <Directory "/var/www/helloworld">
            allow from all
            Options None
            Require all granted
          </Directory>
        </VirtualHost>
      webtext: |
       <html>
         <head><title>Test Site</title>/head>
         <body>
          <h1>Hello, world!</h1>
          <p>The site is up and running</p>
         </body>
       </html>

  tasks:
    - name: Update apt cache
      apt: update_cache=true
    - name: Install necessary packages
      apt:
        name: apache2=2.4.41-4ubuntu3.9

    - name: Install curl, openssl, PyOpenSSL
      apt:
        pkg:
        - curl
        - openssl
        - python3-openssl

    - name: Create a directory for certs
      file:
        path: /etc/apache2/certs
        state: directory
```

```
      mode: 0640

  - name: Generate an openssl key
    openssl_privatekey:
      path: /etc/apache2/certs/apache.key

  - name: Generate an openssl csr
    openssl_csr:
      path: /etc/apache2/certs/apache.csr
      privatekey_path: /etc/apache2/certs/apache.key
      common_name: myhost

  - name: Generate a self-signed openssl certificate
    openssl_certificate:
      path: /etc/apache2/certs/apache.crt
      privatekey_path: /etc/apache2/certs/apache.key
      csr_path: /etc/apache2/certs/apache.csr
      provider: selfsigned

  - name: Configure Hello World virtual host
    copy:
      content: "{{ conftext }}"
      dest: /etc/apache2/sites-available/helloworld.conf
      mode: 0640
      force: yes

  - name: Create the helloworld directory
    file:
      path: /var/www/helloworld
      state: directory
      mode: 0755

  - name: Deploy the Hello World website
    copy:
      content: "{{ webtext }}"
      dest: /var/www/helloworld/index.html
      owner: root
      group: root
      mode: 0644
      force: yes

  - name: Deactivate the default virtualhost
    command: a2dissite 000-default

  - name: Activate the virtualhost
    command: a2ensite helloworld
```

```
      notify:
        - Restart apache

    - name: Activate SSL on Apache
      command: a2enmod ssl
      notify:
        - Restart sshd
        - Restart apache

  handlers:
    - name: Restart sshd
      ansible.builtin.service:
        name: sshd
        state: restarted

    - name: Restart apache
      ansible.builtin.service:
        name: apache2
        state: restarted
```

This playbook configures a website based on the expectations defined earlier in InSpec. To evaluate, you can use Test Kitchen to quickly ensure that the playbook runs without errors, and that the resulting state of the system it configures matches your expectations. Test Kitchen's behavior is defined in a configuration file called `kitchen.yml`. The example below applies the Ansible playbook configurations to an ephemeral Ubuntu 20.04 Vagrant node:

```
---

driver:
  name: vagrant

provisioner:
  hosts: myhost
  name: ansible_playbook
  require_ansible_repo: true
  ansible_verbose: true
  ansible_version: latest
  require_chef_for_busser: false
  playbook: website_https.yml

verifier:
  name: inspec

platforms:
  - name: ubuntu-20.04
```

```
suites:
  - name: default
    verifier:
      inspec_tests:
        - path: tests/website_https_verify.rb
```

The key sections to note in this configuration file are:

- **driver**: How should this ephemeral test instance be launched? This example uses Vagrant, but you could also use Dokken (Docker), cloud instances or other hypervisors.
- **provisioner**: What tool should be used to configure the instance? This example uses the Ansible playbook to configure the node.
- **verifier**: What tool should be used to validate the instance once it's been configured? This example uses Chef InSpec and the `website_https_verify.rb` test profile shown above.

In the above example, running "kitchen test" will take the following actions:

- **Create**: Launch a local Ubuntu 20.04 VM with Vagrant
- **Converge**: Apply the playbook defined in `website_https.yml` to the VM.
- **Verify**: Apply the `website_https_verify.rb` InSpec tests
- **Destroy**: If all of the above actions complete without error, destroy the instance when complete.

If any of your configurations fail in the "converge" step, or any of your InSpec resources return a failure, Test Kitchen will halt its execution and leave the VM running for further inspection. Otherwise, the instance cleans up after itself, and you're ready to apply your playbook to the instances you manage in dev, test or production environments with confidence.

Security Assessment: Detect and Correct Vulnerabilities with InSpec & Ansible

Another area where InSpec can have a profound impact on your organization is in security assessment. New software vulnerabilities are always being discovered, and in order to secure your estate, it's imperative to quickly assess whether your systems are impacted and remediate them accordingly.

Detect and Correct: The POODLE Vulnerability

Several years ago, a vulnerability popularly known as POODLE was added to the national vulnerability database and impacts SSL, the primary protocol for encrypted web traffic. In this vulnerability, SSLv3 was shown to contain a design flaw that allows attackers to obtain clear-text content of ostensibly encrypted data. Therefore, the recommendation is to disable this older SSL protocol and use only Transport Level Security (TLS) connections on modern webservers.

Chef InSpec includes an "ssl" resource that can be used to determine whether or not your servers are configured with SSLv3 support. In the above example, you added resources to the `website_https_verify.rb` profile to verify SSLv3 is not running, and TLS is:

```
describe ssl(port: 443).protocols('ssl3') do
  it { should_not be_enabled }
end

describe ssl(port: 443).protocols('tls1.2') do
  it { should be_enabled }
done
```

As you saw in the previous example, Test Kitchen helped evaluate code updates on temporary infrastructure, but when it comes to vulnerability assessment, it's important to be able to check your live systems. Chef Workstation includes the `inspec` command-line utility, which can be used to directly scan systems you manage over SSH or WinRM. To assess one of your servers, you can run the following command:

```
$ inspec exec test/website_https_verify.rb -t
ssh://myuser@myhost


$ inspec exec test/website_https_verify.rb -t
winrm://Administrator@myhost -P 'mypassword'
```

When you run the above command against one of your servers, you may get back a summary that looks like this, showing your SSL configuration is not correct:

```
Profile: tests from website_https_verify.rb

Version: (not specified)
Target: ssh://myuser@myhost:22
  SSL/TLS on
    Ø myhost:443 with protocol == "ssl3" should not be enabled
    expected SSL/TLS on myhost:443 with protocol == "ssl3" not
    to be enabled
```

As mentioned, the remediation for POODLE is to allow only the TLS protocols for SSL traffic in your webserver's configuration. Using Apache as an example again, a few simple Ansible remediation tasks in a playbook called `poodle_fix.yml` might look like this:

```
…
tasks:
  - name: Fix SSL in Apache
    replace: dest=/etc/apache2/mods-available/ssl.conf
      regexp='^SSLProtocol.*$'
      replace='SSLProtocol -all +TLSv1.2'
    notify:
      - Restart apache2
      - Restart sshd
handlers:
  - name: Restart apache
```

```
    ansible.builtin.service:
      name: apache2
      state: restarted

  - name: Restart sshd
    ansible.builtin.service:
      name: sshd
      state: restarted
```

You can then execute this playbook on one of your hosts similarly to how it was scanned with InSpec:

```
$ ansible-playbook poodle_fix.yml

PLAY [myhosts] ***************************************************

TASK [Gathering Facts]*******************************************

ok: [myhost]

TASK [Fix SSL in Apache] *****************************************
changed: [myhost]

RUNNING HANDLER [Restart sshd]***********************************
changed: [myhost]

RUNNING HANDLER [Restart apache2]********************************
changed: [myhost]

PLAY RECAP *******************************************************
myhost : ok=4 changed=3 unreachable=0 failed=0
```

Updating Apache's SSL configuration and triggering a restart of the *sshd* and *apache2* services, should be sufficient to ensure your system is not vulnerable to POODLE. The operative word, of course, is "should." With configuration management you can put the right configurations in place, but without being able to functionally validate them, your job is only half done.

What if, for example, someone manually re-edited the Apache configuration file to re-enable SSLv3, something you would not detect until some point in the future when you happen to re-run this Ansible playbook on the machine? Because your compliance requirements have been defined as code, validating your newly updated configuration is as simple as re-running the same Chef InSpec scan to ensure SSLv3 is indeed disabled on your system.

```
$ inspec exec website_https_verify.rb -t ssh://myuser@myhost
Version: (not specified)
Target: ssh://myuser@myhost:22
  SSL/TLS on
     ü myhost:443 with protocol == "ssl3" should not be enabled
Test Summary: 1 successful, 0 failures, 0 skipped
```

Success!

Compliance Auditing: Scanning Ansible Environments with InSpec

Scanning Ansible environments with Inspec is functionally very similar to evaluating for security problems. The biggest difference is that compliance frameworks formalize this process around a set of known benchmarks. That said, auditing for compliance requires some extra functionality that hasn't been covered so far. In particular, evaluating compliance will require:

- **Impact Assessment:** A compliance report needs to be able to filter individual results based on their severity for prioritizing remediation.
- **Role-Specific Granularity:** Compliance officers, InfoSec teams and operators each have a role in assessing compliance, but with varying levels of needed detail. High-level summaries and detailed methodologies should both be available and easy to reference.

Chef InSpec *controls* are designed with these concerns in mind so that collecting multiple validations into a Compliance Profile can be used to generate weighted reports with multiple levels of granularity in Chef Automate, the graphical dashboard used to monitor hundreds or thousands of systems at once. Here, simple Chef InSpec resources are included within controls that include a title, description, tags and the actual InSpec resources that do the inspection.

```
control "Ensure_SSH_root_login_is_disabled" do
  title "Ensure SSH root login is disabled"
  desc  "
    The PermitRootLogin parameter specifies if the root user can log in using
    ssh. The default is no.

    Rationale: Disallowing root logins over SSH requires system admins to
    authenticate using their own individual account, then escalating to root
    via sudo or su. This in turn limits opportunity for non-repudiation and
    provides a clear audit trail in the event of a security incident
  "
  tag group: 'SRG-OS-000112'
  tag vulid: 'V-38607'
  tag ruleid: 'SV-50408r1_rule'
  tag severity: 'CAT I'
  tag stigid: 'RHEL-08-000227'
  tag cci: 'CCI-000774'
  tag fixtext: 'PermitRootLogin should not be permitted. The default setting
      in "/etc/ssh/sshd_config" is correct, and can be verified by ensuring
      that the following line appears: PermitRootLogin no'
  impact 1.0
  describe.one do
    describe sshd_config("/etc/ssh/sshd_config") do
      its("PermitRootLogin") { should_not eq('yes') }
    end
    describe package("openssh-server") do
      it { should_not be_installed }
    end
```

```
  end
end
```

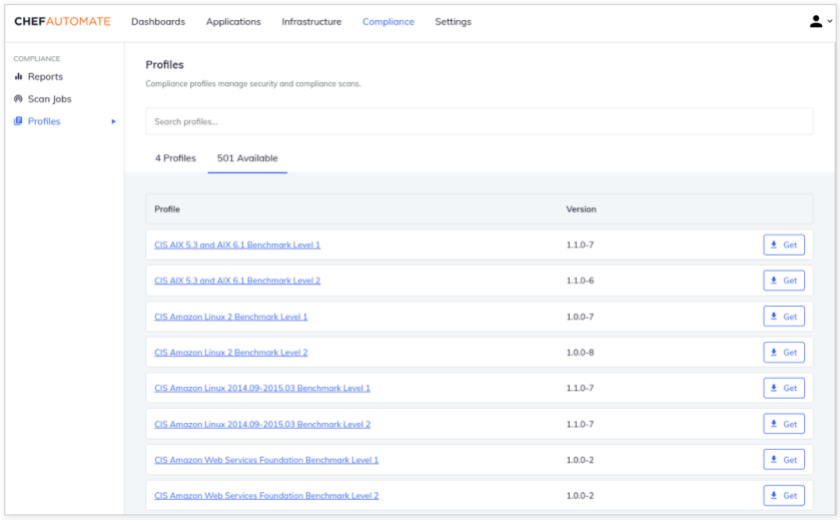
The above example, saved in a file called `ssh_profile.rb`, is an InSpec control, translating a rule from the Red Hat Enterprise Linux 8 DISA STIG benchmarks. It provides examples of some of the extra data necessary for thorough compliance evaluation. The "impact" of a control defines its criticality on a range from 0.0 (minor) to 1.0 (critical). The "title" and "description" provide a human-readable summary of what the control is validating. Each "tag" provides added user-defined metadata, in this case referencing where requirements are defined in the STIG. Finally, "sshd_config" is another example of an InSpec resource, like the "port", "http", and "ssl" InSpec resources covered earlier.

Compliance Reports in Chef Automate

Chef Automate provides a library of more than 500 pre-written Compliance Profiles, as well as dashboards to give you a consolidated view of the compliance of your estate as a whole, filterable by environments, server roles, and audit severity.

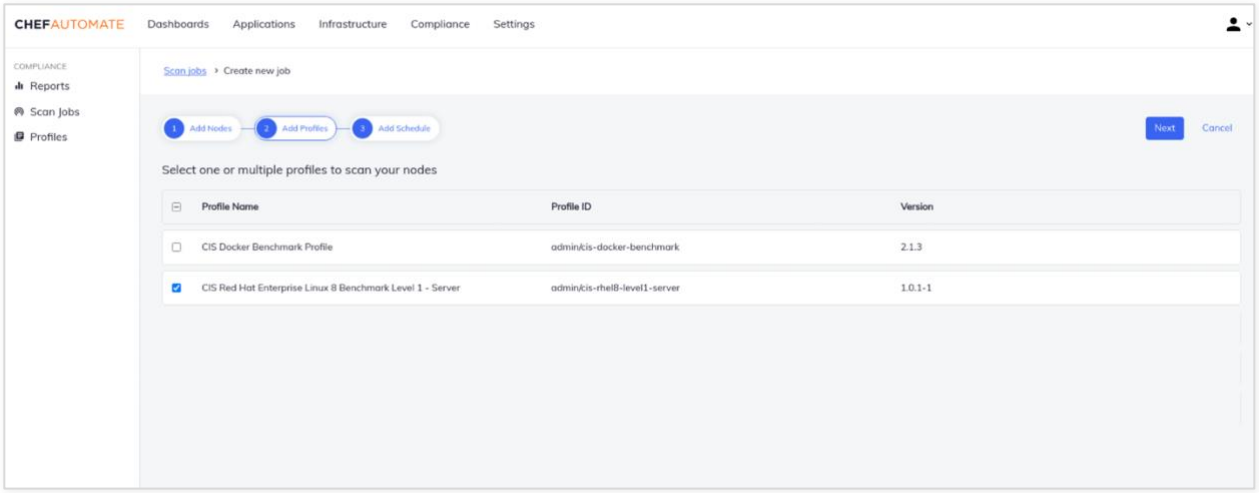
One challenge for Ansible users is that the default method for collecting this data was typically via a specialized Chef Audit Cookbook, which runs InSpec and reports the results to Chef Automate. In a Chef workflow, this cookbook would be pulled from a Chef Server, but for organizations using Ansible, that's not always a feasible option. Thankfully, Chef Compliance does away with the need to use the dedicated Chef Audit cookbook and allows users to include Chef InSpec controls in any cookbook (or Ansible playbook) or as standalone tests, as shown in the examples above. Chef Automate can ingest these profiles and allow users to generate audits by running agentless scans directly from the Automate Server on any node – even if they weren't configured with Chef.

To start, you'll need a Compliance Profile, which is a collection of InSpec controls organized around a specific theme, like the single SSH *control* example shown above. Chef Automate comes pre-loaded with a variety of profiles based on different compliance frameworks and operating systems. You can view all available profiles by clicking **Profiles > Available** tab within Automate's "Compliance" dashboard. Clicking the **Get** button next to any listing will add the profile to your environment.



Clicking **Get** installs the latest version of the profile, which can now be used to start scanning your servers. From there, the **Scan Jobs** link on the left-hand menu can be used to define the nodes you wish to scan, including hostnames, connection protocol (SSH or WinRM), and access credentials. You can add nodes you want to scan in the Automate **Settings** tab under **Node Integrations > Automate**. One or more nodes can be added with IP addresses are domain names, and they don't need to have the Chef Client installed.

Once you have at least one profile and at least one node configured, you can create a "Scan job."



Jobs can be configured for one-time, at-once execution or on a recurring schedule. Simply select all the nodes you'd like to encapsulate in the job, and click "Create job" in the upper right-hand corner.

Once the job has run, the scan job report is available in the dashboard and shows a high-level summary of how your node(s) performed, and how many, if any, of the controls within your profiles failed.

On each node, you can also view a detailed list of each control that was run, with any failed nodes filterable based on their severity. From this same view, you can click on any individual control for more details about its status, and even the raw InSpec source code.

Whether you're managing one server, or thousands, Chef Automate provides a single pane of glass for views into compliance performance across your estate. Within Automate's dashboards, you can view results with whatever level of granularity you require in your role – all without needing to install any agents on your servers.

Summary

To deliver software at high velocity, it's critical to have the means to detect misconfigurations and security flaws consistently and continuously across all the systems you manage. Chef InSpec and Ansible are both designed with automation and repeatability in mind, and together help ensure that you can deliver software quickly, efficiently, and above all, securely. With Chef Automate you can take this process further with a library of pre-written profiles to jump-start your compliance, and a single window into the health of every environment you manage.

Resources

Learning

- Test Expectations with InSpec: https://learn.chef.io/courses/course-v1:chef+Inspec101+Perpetual/about
- Chef Compliance: First Steps with Auditing and Remediation: https://learn.chef.io/courses/course-v1:chef+SECCOM101+Perpetual/about
- Chef Principles Certification Exam: https://learn.chef.io/courses/course-v1:chef+CP101+exam/about?_ga=2.152463609.1126152281.1641392943-2046324149.1631017558

Documentation

- InSpec Docs: https://docs.chef.io/inspec/
- Chef Automate Docs: https://docs.chef.io/automate/

Blogs, Web pages & Webinars

- Chef and Ansible: https://www.chef.io/ansible
- Chef Infra 101: Road to Best Practices: https://www.chef.io/blog/chef-infra-101-the-road-to-best-practices

About Progress

Dedicated to propelling business forward in a technology-driven world, Progress (NASDAQ: PRGS) helps businesses drive faster cycles of innovation, fuel momentum and accelerate their path to success. As the trusted provider of the best products to develop, deploy and manage high- impact applications, Progress enables customers to develop the applications and experiences they need, deploy where and how they want and manage it all safely and securely. Hundreds of thousands of enterprises, including 1,700 software companies and 3.5 million developers, depend on Progress to achieve their goals—with confidence. Learn more at www.progress.com.

facebook.com/getchefdotcom
twitter.com/chef
youtube.com/getchef
linkedin.com/company/chef-software
learn.chef.io
github.com/chef
twitch.tv/chefsoftware

Progress®