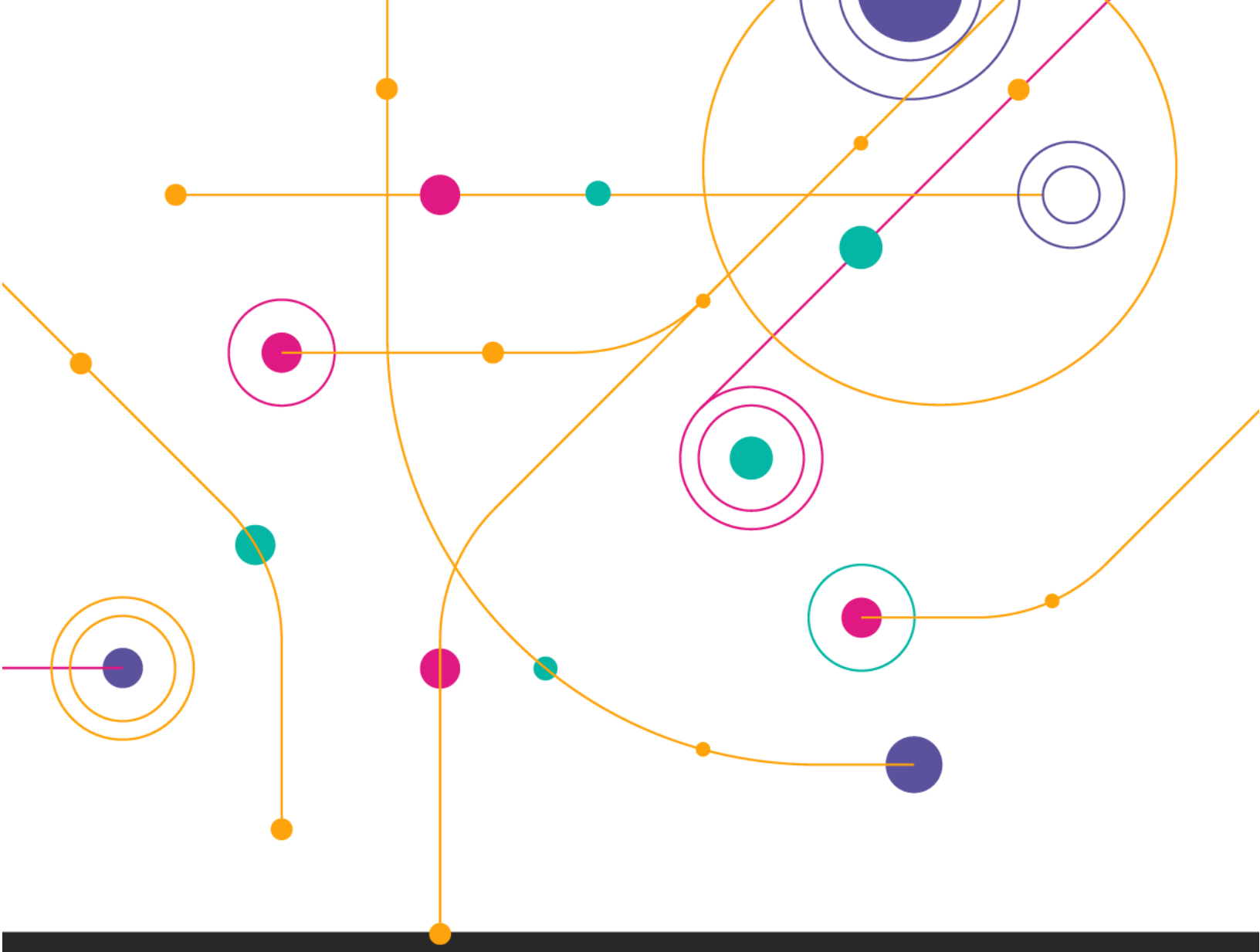


CHEF
Progress

Automatically Deliver Chef:
An Agile Workflow Using Chef and GitLabs

By Stephanie Laingen, DevOps Consultant/Software Engineer
TapHere! Technology

TapHere! is a Proud Progress Chef Partner

TapHere! TECHNOLOGY

Guide Summary:

This is a companion guide to the ChefConf '21 Session "Automatically Deliver Chef: An Agile Workflow". The guide shows how to create a reliable workflow that supports agile development, standardizes Chef coding practices, and automatically delivers changes to Chef managed nodes using GitLab pipelines

Table of Contents

Overview

No matter what code you are writing, a reliable CI/CD (Continuous Integration/Continuous Delivery) strategy is essential to agile development. If the goal is "early and continuous delivery of valuable software" and to "deliver working software frequently", to quote the Agile Manifesto, then a dependable workflow to get new code quickly available in a live environment for real world testing and later into a production environment is definitely pretty important.

This tutorial will walk you through the steps of creating a CI/CD pipeline that allows newly written Chef code to be tested within minutes of a merge. This allows developers to see immediately how their code will act in a live environment that closely mirrors production with no effort on their part. Cutting out the need for complicated or tedious deployment steps, developers can spend more time testing and developing their code instead of running repetitive commands in their terminals. By capitalizing on basic concepts like Git branches, versioning, and Chef Policyfiles, this workflow is elegantly simple and beginner friendly.

We will start by evaluating how to get cookbook code to a live testing node (what we will call the dev node) manually, and then automate steps that can be completed easily by a machine rather than a developer. This will streamline the process of deploying code, and require human interactions only at decision points, like a merge request. This ensures that developers are still in the loop, and that the cookbook code is verified by both human decisions and automated methods.

Assumptions

This tutorial assumes that you have experience with the following development tools:

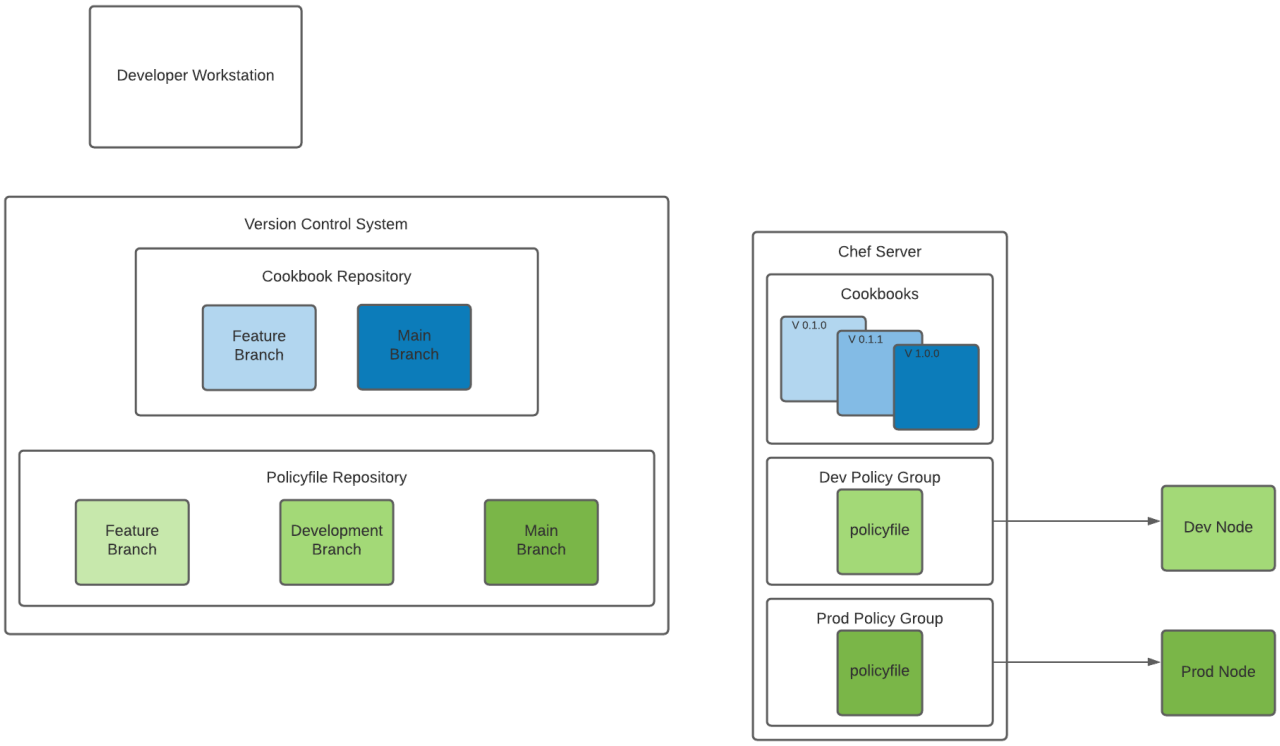
- Git
- A Git repository hosting service with a CI/CD capability (we use GitLab)
- Pull Requests/Merge Requests

This tutorial assumes that you have beginner experience with the following Chef products/concepts:

- Chef Infra (be familiar with writing a basic cookbook)
- Chef Policyfiles and Policy Groups
- Chef Workstation (Test Kitchen, Cookstyle, and other command-line tools)
- Chef Infra Server (create a chef server and bootstrap nodes to it)
- Chef Automate (not required but will be used for visual reference in this tutorial)

Additional Guides that Maybe Useful:
- Accelerate Test-driven Development with Chef Workstation and Test Kitchen
- Getting Started with Chef Automate: Install and Configure Chef Automate and Infra Server
- Chef Infra Best Practices: Using Chef Infra Policyfiles (YouTube)

The starting point of this tutorial is described in the following diagram:

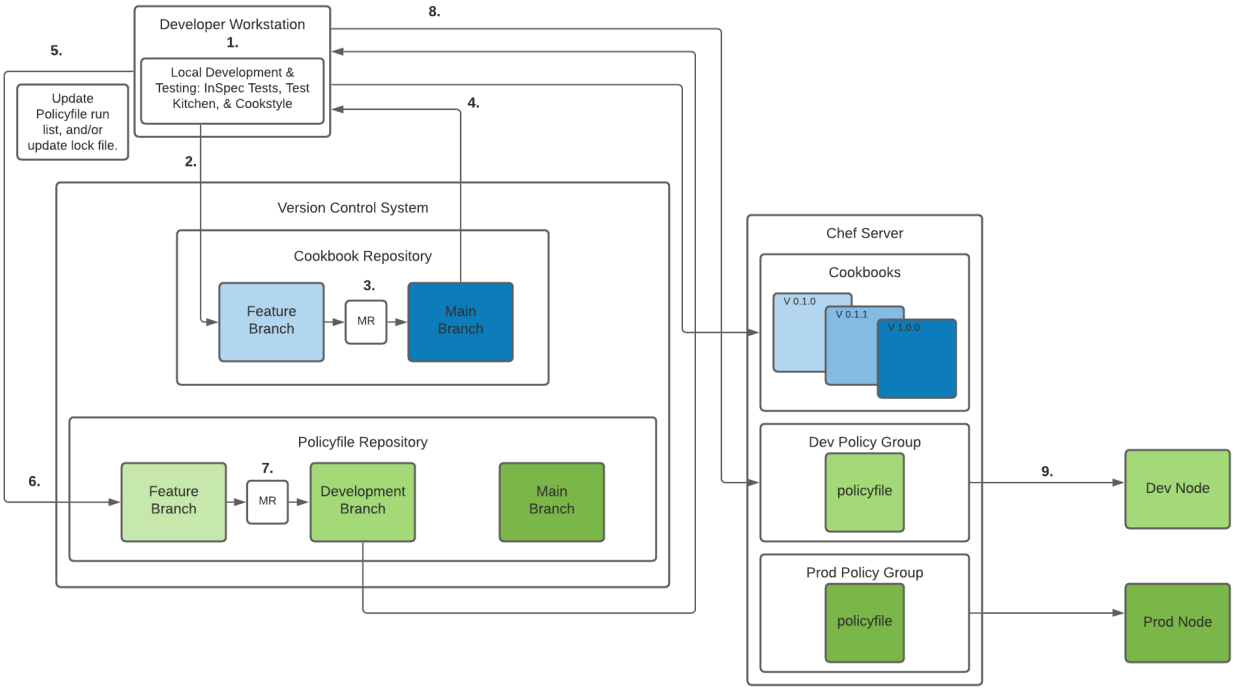


Further description of the elements above:

- **Developer Workstation** - A computer with Chef Workstation installed on it set up with credentials to speak to a Chef Infra/Automate Server.
- **Cookbook Repository** - A repository hosted on GitLab containing a cookbook with a default recipe that performs an action.
- **Policyfile Repository** - A repository hosted on GitLab containing a policyfile that contains our cookbook from the cookbook repo in its run list.
- **Chef Server** - A server with the Chef Infra Server installed on it. This will store the cookbooks and various policyfiles (uploaded by policy group). Bootstrapped nodes check into this service on an interval.
- **Dev Node** - A computer that has been bootstrapped to the Chef Infra Server using a policyfile and the dev policy group. This means that when the node checks in, it will run the cookbooks in the run list of the policyfile it is attached to that has been uploaded to the dev policy group.
- **Prod Node** - A computer that has been bootstrapped to the Chef Infra Server using a policyfile and the prod policy group. This means that when the node checks in, it will run the cookbooks in the run list of the policyfile it is attached to that has been uploaded to the prod policy group.

Make a Change in a Cookbook to be Tested on the Dev Node

Without a CI/CD workflow in place, getting changes in your cookbook code on the dev node for testing requires many steps. The diagram below may look complicated, because it is! Our goal is to simplify and automate this process using pipelines so that developers do not have to worry about performing these steps every time they make a change.



Steps Broken Down:

- **Local Development & Testing** - Move to a feature branch, make code change, and begin local testing. Write tests that describe any added functionality, run cookstyle to lint code, run test kitchen (kitchen test) to test code, and update the version of the cookbook accordingly.

- **Push Cookbook Code on Feature Branch to GitLab** - Run in terminal on developer workstation: git push origin feature_branch.

- **Merge Request & Peer Approval** - Create a merge request for the feature branch into the development branch. Peers can test the code the code locally using kitchen test and inspect the code using cookstyle. If the changes are approved by the other developers, the feature branch is merged into the development branch. The same process can be performed to get the development branch merged into the master branch.

- **Upload Cookbook to Chef Server** - Once the new code has been merged into the master branch, git pull the master branch to the developer workstation to run the knife cookbook upload command to upload the cookbook to the Chef Server.

- **Update Policyfiles** – Update the version of the cookbook that is run in the policyfile to the new version just uploaded to the server. Manually run chef install and chef update on the policyfiles to generate new lock files. Commit changes to GitLab on a feature branch in the policyfiles repo.

    - **Push Policyfile Code on Feature Branch to GitLab** - Run in terminal on developer workstation: git push origin feature_branch.
    - **Merge Request & Peer Approval** - Create a merge request in the Policyfiles Repository for the feature branch into the development branch. When approved, the development branch will contain the update policyfile lock files from the feature branch.
    - **Push Policyfile to Dev Policy Group** - Once the new code has been merged into the development branch, git pull the development branch to the developer workstation to run the chef push dev policy-file command to upload the policyfile to the dev policy group on the Chef Server.
    - **Dev Node Runs Cookbook on Next Check-In** - At the next check-in, the dev node will automatically run the updated cookbook code because it is attached to the dev policy group.

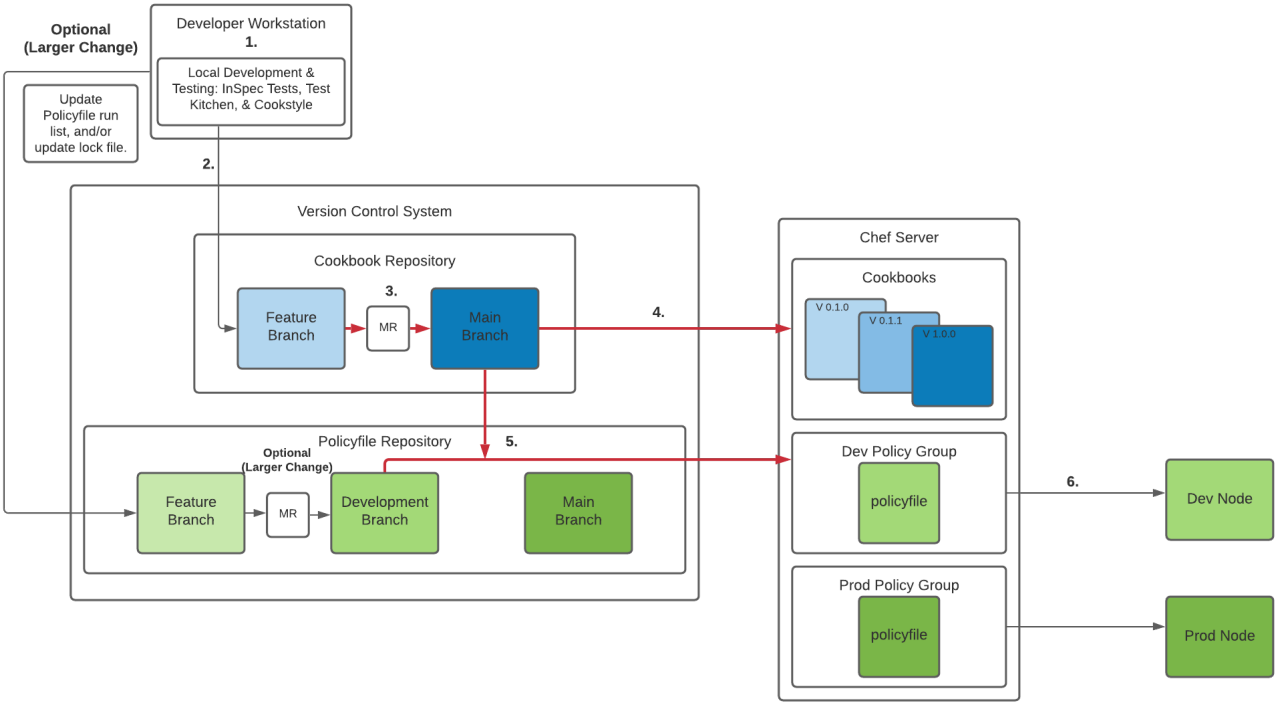
    Additional Resources:

    - Chef Infra Best Practices: Automate Cookbook Testing with GitHub Actions
    - Chef DevRel Twitch Stream

Automated Steps to Deploy Code to Dev Node - Just 1 Merge!

By enabling a machine to run the commands that a human would typically have to run from their developer workstation, we can save valuable developer time and deploy code quickly. GitLab enables pipelines, a CI/CD solution that uses YAML documents that describes steps to be run by a computer when certain actions occur within a repo, like merge request creation or approval. GitHub's version of this is called GitHub actions and Azure has Azure pipelines. This tutorial will utilize GitLab pipelines, but these actions can be performed in another CI/CD system.

Below are the steps needed to make a change in a cookbook to be tested on the dev node when using GitLab pipelines:



Steps Broken Down:

1. **Local Development & Testing** - Move to a feature branch, make code change, and begin local testing. Write tests that describe any added functionality, run cookstyle to lint code, run test kitchen to test code, and update the version of the cookbook accordingly.

2. **Push Cookbook Code on Feature Branch to GitLab** - Run in terminal on developer workstation: git push origin feature_branch.

3. **Merge Request & Peer Approval** - Create a merge request for the feature branch into the development branch. Peers do not need to test the code locally because *the pipeline in the cookbook repository will automatically run cookstyle and test kitchen.* If the pipeline run is successful and the changes are approved by the other developers, the feature branch is merged into the development branch. The same process can be performed to get the development branch merged into the master branch.

4. **Pipeline Cookbook Upload & Trigger Policyfile Pipeline** - *The pipeline file in the cookbook repository will automatically upload the cookbook to the Chef Server upon a successful merge.* This means that the developer does not have to pull down the master branch manually and run commands in their terminal. *The cookbook repository pipeline will then trigger the pipeline to run in the policyfile repository on the development branch.*

5. **Policyfile Pipeline runs on the Development Branch** - *The policyfile pipeline will run the chef install and chef update command on all of the policyfiles in the development branch, updating their lock files to pull in the new version of the cookbook to be tested. The pipeline will commit these changes to the development branch and use the chef push command to push the new lock files to the dev policy group on the chef server.*

6. **Dev Node Runs Cookbook on Next Check-In** - At the next check-in, the dev node will automatically run the updated cookbook code because it is attached to the dev policy group.

Use Loose Version Pinning

In order to ensure that step 5 of the workflow runs properly, ensure that policyfiles refer to cookbooks with a loose version pinning. If a policyfile has a cookbook included with a specific version, running chef install and chef update on that policyfile will not cache a new version of the cookbook. This is what a strict version pinning looks like:

```
cookbook 'tar', '0.1.0'
```

For the new version of the cookbook to be included in the run list, the policyfile must include the cookbook with a loose version pinning that will allow for a range of versions to be valid. One way to do this is to use the ~ character to indicate a range. In this example, we will set a range from 0.1.0 to 0.2.0, which will allow for small bug fixes (e.g. 0.1.1) to be included in the range. This means that if a new cookbook version becomes available on the chef server that fits into the range, it will be cached and included in the run list.

```
cookbook 'tar', '~0.1.0'
```

How to Set Up GitLab Pipelines

A pipeline is just a specific set of commands that is executed by some machine. Using this simplified definition, 2 parts are necessary to enable pipelines:

- **The machine** - For the pipeline to run any command, there must be a computer set up on which to run the command. GitLab calls this the **GitLab Runner**.

- **The set of commands** - GitLab recognizes the file named **.gitlab-ci.yml** in the repository as the set of commands that the pipeline will run. Each task that you want the pipeline to run is a job, and the **.gitlab-ci.yml** file contains all of the information about when a job runs, and what it does.

Setting Up the Runner

The pipelines will run commands that a developer would typically run in their developer workstation. The developer workstation is set up with credentials to access the Chef Server and has Chef Workstation installed on it so that chef commands are available to be used in the terminal. To run these commands in the GitLab Runner, it will also need Chef Workstation installed and credentials to access the Chef Server.

1. **Install the GitLab Runner and register it to your GitLab Instance.** Start with a computer that has access to the same network as your GitLab server. Steps can be found here to install the GitLab Runner on the machine register it with your GitLab Server. The runner that we are using is run on a Linux machine that uses the shell executor.

2. **Install Chef Workstation on the runner.** Installation steps here. Ensure that you create a .chef folder in the /home/gitlab-runner/ directory so that the gitlab-runner user that runs the pipeline jobs has access to it. We will eventually put a config.rb file into this folder to hold the credentials that will allow the GitLab Runner to connect to the Chef Server.

3. **Create credentials for the GitLab Runner on the Chef Server and store them in the runner.** The documentation on creating users on a Chef Server and instructions for setting up credentials are helpful resources, but these are the exact steps that we took to get the GitLab Runner credentials configured:

    - SSH into the Chef Server.
    - Use the chef-server-ctl user-create command to create a user named grunner. You will want to save the pem for the GitLab Runner user to a file.
    - Use the chef-server-ctl org-user-add command to add the GitLab Runner user to whatever organization your other users are attached to.
    - Store the grunner.pem file in the .chef directory that you created in the /home/gitlab-runner/.chef directory.

- Create a config.rb file in the /home/gitlab-runner/.chef directory with the following configuration:

```
client_name 'grunner'
node_name 'grunner'
client_key '/home/gitlab-runner/.chef/grunner.pem'
chef_server_url 'chef_server_url/organizations/org_name'
ssl_verify_mode :verify_none
```

The Code

To create a pipeline, a file name **.gitlab-ci.yml** must be added to both the Cookbook and Policyfile repositories. Each task that you want the pipeline to run is a job, and jobs are run in stages. While the docs have more detailed information, for the purposes of the jobs we want to run, we will be using the default stage names, test and deploy, and will be using the following format to write our jobs:

```
job-name:

  only: #used for jobs that only run on a specific branch

    refs:
      - branch-name
  stage: stage-name
  script:
    - command 1
    - command 2
```

The jobs that need to be run in the Cookbook repository pipeline:

→Run cookstyle, the linting tool on every branch, every time there is a change. (Step 3 of the workflow)

```
cookstyle:
  stage: test
  script:
    - echo "Running Cookstyle"
    - cookstyle
```

→Run test kitchen using the kitchen test command on every branch, every time there is a change. (Step 3 of the workflow)

```
test-kitchen:
  stage: test
  script:
    - echo "Running Test Kitchen"
    - CHEF_LICENSE=accept kitchen test
```

→Use the knife cookbook upload command to upload the cookbook to the Chef Server only when there is a change to the master branch. (Step 4 of the workflow)

```
deploy-prod:

  only:
    refs:
      - master
  stage: deploy
  script:
    - echo "Push Cookbook to Chef Server"
    - knife cookbook upload $CI_PROJECT_NAME --cookbook-path .. --profile
default
```

When a cookbook is successfully uploaded, trigger the pipeline in the Policyfile repository so that the policyfiles can use the new version of the cookbook that is now available on the Chef Server. (Step 4 of the workflow)

```
trigger-policyfile-update:
  only:
    refs:
      - master
  stage: deploy
  trigger:
  project: gitlab-username/policyfiles
  branch: development
```

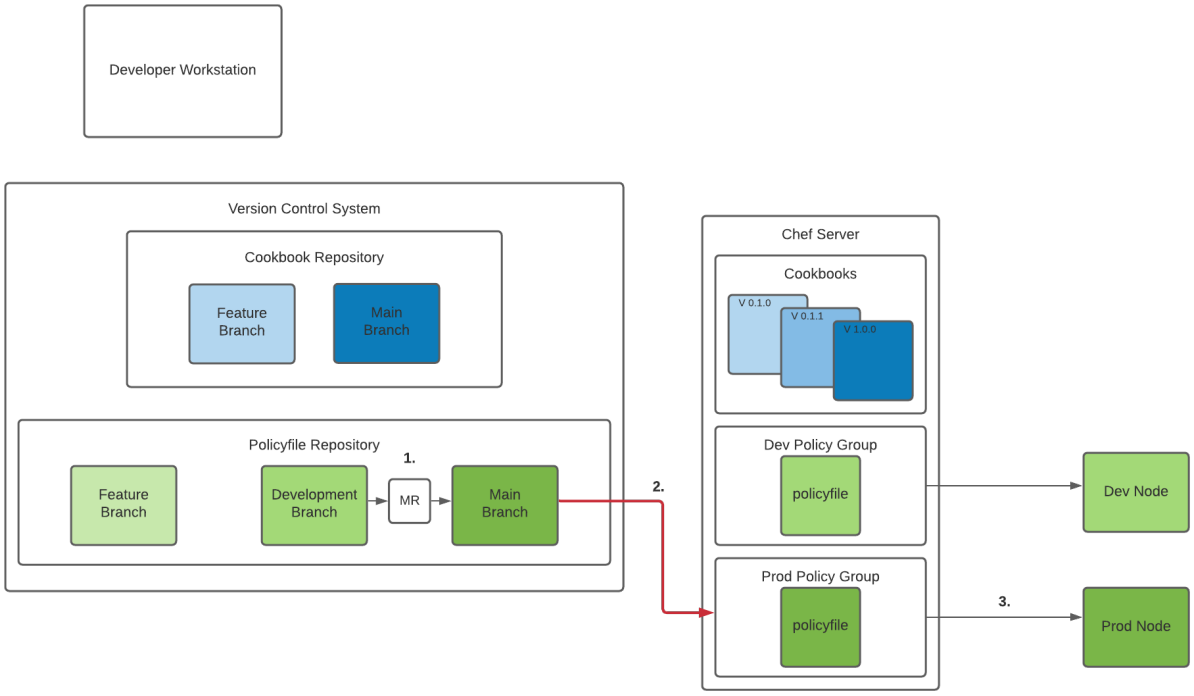
The jobs that need to be run in the Policyfile repository pipeline:

Run chef install and chef update to update the policyfile lock files on the development branch when there is a change. Commit the updated lock files to git and run chef push dev on every policyfile to upload them to the Chef Server. (Step 5 of the workflow)

```
push-dev-policyfile:
  only:
    refs:
      - development
  stage: deploy
  script:
    - echo "Update policyfiles"
    - git config user.email "gitlabrunner@example.com"
    - git config user.name "CI Pipeline"
    - for FILE in $PWD/*; do if [[ $FILE =~ \.rb$ ]]; then chef install
$FILE -c /home/gitlab-runner/.chef/config.rb ; fi; done
    - for FILE in $PWD/*; do if [[ $FILE =~ \.rb$ ]]; then chef update
$FILE -c /home/gitlab-runner/.chef/config.rb ; fi; done
    - for FILE in $PWD/*; do if [[ $FILE =~ \.rb$ ]]; then chef push dev
$FILE -c /home/gitlab-runner/.chef/config.rb ; fi; done
    - git add .
    - git commit -m "this is a pipeline commit [skip ci]" || true
    - git push http://${PUSH_USER_NAME}:${ACCESS_TOKEN}@ps-
gitlab.taphere.com/slaingen/policyfiles.git HEAD:development || true
```

If the Code Works on the Dev Node – Get it the Prod Node with 1 Merge!

Once the dev node has run successfully, you can confidently run the new version of the cookbook on the production node. The prod node runs whatever cookbooks are in the run list of the policyfile in the prod policy group. This means that in order to get the cookbook that is running on the dev node onto the prod node, we need to upload the policyfile that is currently in the dev policy group and to the prod policy group. Pipelines can help automate this action, making the transition from dev to prod just 3 steps, only 1 of which involving human interaction.



Steps Broken Down:

1. **Merge Request and Peer Approval** - Create a merge request to merge the policyfile with the updated run list and lock file on the development branch into the master branch. Peers can review the performance of the dev node and approve the request to merge into master.

2. **Policyfile Pipeline runs on the Master Branch** - Once the updated policyfiles are merged into master, *the policyfile pipeline will run the chef install command on all of the policyfiles in the development branch, ensuring that the proper cookbook versions are cached, and use the chef push command to push the new lock files to the dev policy group on the chef server.*

3. **Prod Node Runs Cookbook on Next Check-In** - At the next check-in, the prod node will automatically run the updated cookbook code because it is in the run list of the policyfile that is attached to the prod policy group.

Pipeline Code:

When there is a change in the master branch, run `chef install` to cache the cookbook data and run `chef push prod` on every policyfile to upload them to the Chef Server. This code should be added to the .gitlab-ci.yml file in the Policyfile repository.

```
push-prod-policyfile:

  only:

    refs:

      - master

  stage: deploy

  script:

    - echo "Push Cookbook to Chef Server"

    - for FILE in $PWD/*; do if [[ $FILE =~ \.rb$ ]]; then chef install $FILE -c
/home/gitlab-runner/.chef/config.rb ; fi; done

    - for FILE in $PWD/*; do if [[ $FILE =~ \.rb$ ]]; then chef push prod $FILE -c
/home/gitlab-runner/.chef/config.rb ; fi; done
```

Conclusion

In this tutorial, we took a process and analyzed each step of it. By recognizing steps that require human interaction that are not decision points, we were able to identify pieces of work that could be done by a machine, freeing up valuable human time for more complex tasks. While the method of automation that we used was GitLab Pipelines, there are many other automation tools that can achieve the same result. This workflow is a dependable way to deploy Chef code to managed nodes efficiently and with human interaction at major decision points (accepting a merge request).

About the Author:

Stephanie Laingen

As a Computer Scientist and DevSecOps Consultant, Stephanie Laingen has a passion for problem solving. After completing her Computer Science degree at Yale, she joined TapHere! Technology, a small team of highly skilled engineers that serves in the federal sector. As a consultant, Stephanie architects custom solutions, develops workflows, and provides bespoke education to meet the personalized needs of each customer, enabling agile and sustainable practices throughout.

TapHere! Technology, LLC (TapHere) is a Progress Chef delivery partner and dedicated to exceeding requirements and increasing efficiency for our customers by providing innovative solutions. Headquarter location 10440 Balls Ford RD. #250, Manassas VA 20109.

ABOUT CHEF AND PROGESS

Progress (NASDAQ: PRGS) provides the best products to develop, deploy and manage high-impact business applications. Acquired in October 2020, Chef extends Progress offerings in DevOps and DevSecOps, with market-leading, modern infrastructure, compliance, and application by automation. With Progress, you can accelerate the creation and delivery of strategic business applications, automate the process which you configure, deploy and scale those apps, and make your critical data and content more accessible and secure— leading to competitive differentiation and business success. Over 1,700 independent software vendors, 100,000+ enterprise customers, and a three-million-strong developer community rely on Progress to power their applications. Learn about Progress at www.progress.com or +1-800-477-6473.

http://www.chef.io