1. DevOps

A hands-on DevOps course.

This course will

- Discuss DevOps,
- Have you spin up a DevOps toolchain and development environment, and then
- Author two applications and their accompanying pipelines, the first a continuous integration (CI) and the second a continuous delivery (CD) pipeline.

We will be spending most of the course hands-on working with the tools and in the Unix command line making CI/CD happen, so as to grow an understanding of how DevOps actually works.

Don't fixate on the tools used, nor the apps we develop in the course of learning how and why. How and why is far more important. This course like DevOps is not about tools although we'll be using them. You'll spend far more time writing code. (Or at least cutting-and-pasting code.)

2. Author

- Michael Joseph Walsh mjwalsh@mitre.org, walsh@nemonik.com

3. Acknowledgments

- Thank you to Walter Hiranpat for running through course material prior to me teaching it.
- Thank you to Eric McCann assisting Walter and I work through Vagrant install issues on Microsoft Windows.
- Thank you to my first class participants for "beta" testing my class.
- Thank you to Jonathan Thomas for TA'ing my second class.

4. Copyright and License

5. TOC

7. Prerequisites

The following skills would be useful in following along, but aren't strictly necessary.

What you should bring:

- Managing Linux or Unix-like systems as we will living largely within the terminal.
- A basic understanding of Vagrant, Docker, and Ansible would be helpful.
- But mostly being a software engineer.

What's on the classroom computer:

- The latest version of VirtualBox is expected to be installed.

6. DevOps unpacked

6.1. What is DevOps?

DevOps (a clipped compound of the words "development" and "operations") is a software development methodology with an emphasis on a more reliable release pipeline, automation, and stronger collaboration across all stakeholders with the goal of deploying working functionality into the hands of users (i.e., production) faster.

Yeah, that's the formal definition. I've grown to prefer the axiom:

You are what you code.

For example,

1. If a developer or software engineer, you're at least coding the application, its build automation, unit tests, and CI/CD automation.
2. If a tester, you're coding the rest of the test automation.
3. If a security engineer, the compliance and policy test automation.
4. If ops, the deployment and infrastructure configuration management automation.

6.2. What DevOps is not.

DevOps will not stop all bugs nor all vulnerabilities from making it into production, but that's not really the point.

6.3. DevOps is really about

DevOps point is about providing the culture and release pipeline that once a bug or vulnerability is discovered for the issue to be quickly remediated and return functionality back to the user.

6.4. How does it related to the Agile?

Agile Software Development is an umbrella term for a set of methods and practices based on the values and principles expressed in the Agile Manifesto.

Agile Software Development shares the same goal, but DevOps extends Agile methods and practices by adding communication and collaboration between

- development,
- quality assurance, and
- technology operations
- to ensure software systems are delivered in a rapid, reliable, low-risk manner.

For Agile, solutions evolve through collaboration between self-organizing, cross-functional teams utilizing the appropriate practices for their context.

Again, in DevOps everyone is developing software, so it is my view DevOps builds on Agile.

6.5. How do they differ?

While Agile Software Development methods encourage collaboration among the cross-functional teams, the focus in DevOps is on the

- inclusion of analysis,
- design,
- development, and
- quality assurance functionaries as stakeholders into the development effort.

6.6. Why?

In Agile Software Development, there is rarely an integration of these individuals outside the immediate application development team with members of technology operations (e.g., network engineers administrators, testers, cyber security engineers.)

6.7. What are the principles of DevOps?

As DevOps matures, several principles have emerged, namely the necessity for product teams to:

- Apply holistic thinking to solve problems,
- Develop and test against production-like environments,
- Deploy with repeatable, and reliable processes,
- Remove the drudgery through automation,
- Validate and monitor operational quality, and
- Provide rapid, automated feedback to the stakeholders

6.8. How is this achieved?

Achieved through Continuous Integration (CI) and Continuous Delivery (CD) often conflated into CI/CD.

This is what is commonly (albeit mistakenly) thought to be the totality of DevOps.

6.9. What is Continuous Integration (CI)?

It is an Agile software development practice associated where members of a product team frequently integrate their work in order to detect integration issues as quickly as possible thereby shifting discovery of issues "left" (i.e., early) in the software release.

6.10. How?

Each integration is orchestrated through a CI service/orchestrator (e.g., Jenkins CI, Drone CI, Concourse CI) that essentially assembles a build, runs unit and integration tests every time a predetermined trigger has been met; and then reports with immediate feedback.

6.11. CI best practices

6.11.1. Utilize a Configuration Management System

For the software's source code, where the mainline (i.e., master branch) is the most the most recent working version, past releases held in branches, and new features not yet merged into the mainline branch being worked on their own branches.

6.11.2. Automate the build

By accompanying build automation (e.g., Gradle, Apache Maven, Make) alongside the source code.

6.11.3. Employ one or more CI services/orchestrators

Perform source code analysis via automate formal code inspection and assessment.

6.11.4. Make builds self-testing

In other words, ingrain testing by including unit and integration tests (e.g., Spock, JUnit, Mockito, SOAPUI, go package *Testing*) with the source code so as to be executed by the build automation to be execute by the CI service.

6.11.5. Never commit broken

Or untested source code to the CMS mainline or otherwise risk breaking a build.

6.11.6. Developers are expected to pre-flight new code

Prior to committing source code in their own workspace.

6.11.7. The CI service/orchestrator provides feedback

On the success or fail of a build integration to all stakeholders.

6.12. What is Continuous Delivery?

It is a software development practice of providing a rapid, reliable, low-risk product delivery achieved through automating all facets of building, testing, and deploying software.

6.12.1. Extends Continuous Integration (CI)

With additional stages/steps aimed to provide ongoing validation that a newly assembled software build meets all desired requirements and thereby is releasable.

6.12.2. Consistency

Delivery of applications into production is achieved through individual repeatable pipelines of ingrained system configuration management and testing

6.13. But wait. What's a pipeline?

A pipeline automates the various stages/steps (e.g., Static Application Security Testing (SAST), build, unit testing, Dynamic Application Security Testing (DAST), secure configuration acceptance compliance, integration, function and non-functional testing, delivery, and deployment) to enforce quality conformance.

6.14. But really why do we code err. automate?

In 2001, I think Larry Wall in his1st edition of Programming Perl book put it best with "We will encourage you to develop the three great virtues of a programmer:

laziness,

impatience, and

hubris."

The second edition of the same book provided definitions for these terms

6.14.1. Laziness

**he quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labor-saving programs that other people will find useful, and document what you wrote so you don't have to answer so many questions about it. Hence, the first great virtue of a programmer.* (p.609)

6.14.2. Impatience

The anger you feel when the computer is being lazy. This makes you write programs that don't just react to your needs, but actually anticipate them. Or at least pretend to. Hence, the second great virtue of a programmer. (p.608)

6.14.3. Hubris

Excessive pride, the sort of thing Zeus zaps you for. Also, the quality that makes you write (and maintain) programs that other people won't want to say bad things about. Hence, the third great virtue of a programmer. (p.607)

6.14.4. But really we automate for

- Faster, coordinated, repeatable, and therefore more reliable deployments.
- Discover bugs sooner. Shifting their discovery left in the process.
- To accelerates the feedback loop between Dev and Ops.
- Reduce tribal knowledge, where one group or person holds the keys to how things get done. Yep, this is about making us all replaceable.
- Reduce shadow IT (i.e., hardware or software within an enterprise that is not supported by IT. Just waiting for its day to explode.)

6.15. How is a pipeline manifested?

Each delivery pipeline is manifested as **Pipeline as Code** (i.e., software automation) accompanying software's source code in its version control repository.

6.16. What underlines all of this?

I'd argue, *a ubiquitous access to shared pools of configurable system resources and higher-level services that can be rapidly provisioned with minimal management effort* (i.e., cloud services) without CI/CD (the repeated practices of DevOps) will struggle.

6.17. Monitoring

Once deployed, the work is done, right?

A development team's work is not complete once a product leaves CI/CD and enters production; especially, under DevOps where the development teams include members of operations.

6.17.1. The most important metric

Working software is the *primary*, but not the only, measure of progress. The key to successful DevOps is knowing how well the methodology and the software it produces are performing.

6.17.2. An Understanding of Performance

Is achieved by collecting and analyzing data produced by environments used for CI/CD and production.

6.17.3. Establish a baseline

To determine what the baseline performance is so that improvements can be gauged and anomalies detected.

6.17.4. Set Reaction Thresholds

Agile methods and practices should formulate and prioritize reactions weighting factors, such as, the frequency at which an anomaly arises and who is impacted.

6.17.5. Reacting

A reaction could be as simple as operations instructing users through training to not do something that triggers the anomaly, or more ideally, result in an issue being entered into the product's backlog culminating in the development team delivering a fix into production.

6.17.6. Gaps in CI/CD

Monitoring will also inform development teams of gaps in CI/CD resulting in additional testing for the issue that triggered the necessity for the improvement.

6.17.7. Eliminating Waste

Further, monitoring may result in the re-scoping of requirements, re-prioritizing of a backlog, or the deprecation of un-used features.

6.18. Crawl, Walk, Run

6.18.1. Ultimately, DevOps is Goal

- With DevOps one does not simply hit the ground running.
- One must first crawl, walk, and then ultimately run as you embrace the necessary culture change, methods, and repeated practices.
- Collaboration and automation are expected continually improve so to achieve more frequent and more reliable releases.

7. Reading list

AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis
William J. Brown, Raphael C. Malveau, Hays W. "Skip" McCormick, Thomas J. Mowbray
ISBN: 978-0-471-19713-3
Apr 1998

Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Addison-Wesley Signature Series (Fowler))

David Farley and Jez Humble
ISBN-13: 978-0321601919
August 2010

The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations
Gene Kim Jez Humble, Patrick Debois, John Willis
ISBN-13: 978-1942788003
October 2016

Release It!: Design and Deploy Production-Ready Software 2nd Edition
Michael T. Nygard
ISBN-13: 978-1680502398
January 18, 2018

8. Now the hands-on part

8.1. Configuring environmental variables

On Mac OS X

```
export proxy="http://gatekeeper.mitre.org:80"
export http_proxy="http://gatekeeper.mitre.org:80"
export https_proxy="http://gatekeeper.mitre.org:80"
export no_proxy=127.0.0.1,localhost,.mitre.org,.local,192.168.0.10,192.168.0.11
export VAGRANT_ALLOW_PLUGIN_SOURCE_ERRORS=0
```

On Windows

1. In the Windows taskbar, enter `edit the systems environment variables` into `Search Windows` and select the icon with the corresponding name.
2. On the `Advanced` tab select `Environment Variables...` button.
3. In `Environment Variables` windows that opens, under `User variables for...` press `New ...` to open a `New User Variable` window, enter `proxy` for `variable name:` and `variable value:` for each pair in the table below

Variable Name	Value
proxy	http://gatekeeper.mitre.org:80
http_proxy	http://gatekeeper.mitre.org:80
https_proxy	http://gatekeeper.mitre.org:80
no_proxy	127.0.0.1,localhost,.mitre.org,.local,192.168.0.10,192.168.0.11
VAGRANT_ALLOW_PLUGIN_SOURCE_ERRORS	0

8.2. VirtualBox

VirtualBox is a general-purpose full virtualizer for x86 hardware, targeted at server, desktop and embedded use.

8.2.1. Installing VirtualBox

For the class, it is assumed VirtualBox is installed, but below are the instructions for installing it on Windows 10.

1. Open your browser to https://www.virtualbox.org/wiki/Downloads
2. Click `Windows hosts` link under `VirtualBox 5.2.6 platform packages`.
3. Find and click the installer to install.

Then turn for Windows 10 turn off Hyper-V

1. Click Windows `Start` and then type `turn Windows features on or off` into the search bar.
2. Select the icon with the corresponding name.
3. This will open the `Windows Features` page and then unselect the `Hyper-V` checkbox if it is enabled and then click `Okay`.

The same site has the Mac OS X download. The install is less involved.

8.3. Git Bash

Is `git` packaged for Windows with bash (a command-line shell) and a collection of other, separate *NIX utilities, such as, `ssh`, `scp`, `cat`, `find` and others compiled for Windows.

8.3.1. Installing Git Bash

If you are on Windows or on a lab computer, you'll need to install `git`.

1. Download from https://git-scm.com/download/win
2. Click the installer.
3. Click `next` until you reach the `Configuring the line ending conversions` page select `Checkout as, commit Unix-style line endings`.
4. Then `next`, `next`, `next` ...
5. On the Windows task bar, enter `git` into `Search Windows` then select `Git Bash`. Use `Git Bash` instead of `Command` or `Powershell`.

8.4. Retrieve course material

If you are reading this on paper and have nothing else, you only have a small portion of the class material. You will need to download the class project containing all the automation to spin up a DevOps toolchain and development, etc.

In a shell, for the purposes of the class, this means in `Git Bash`, clone the project from https://gitlab.mitre.org/mjwalsh/hands-on-DevOps.git via git like so:

```
git -c http.sslVerify=false clone https://gitlab.mitre.org/mjwalsh/hands-on-DevOps.git
```

Output will resemble:

```
→ git -c http.sslVerify=false clone https://gitlab.mitre.org/mjwalsh/hands-on-DevOps.git
Cloning into 'hands-on-DevOps'...
remote: Counting objects: 742, done.
remote: Compressing objects: 100% (30/30), done.
remote: Total 742 (delta 12), reused 0 (delta 0)
Receiving objects: 100% (742/742), 233.15 MiB | 14.82 MiB/s, done.
Resolving deltas: 100% (285/285), done.
```

8.5. Vagrant

This class uses Vagrant, a command line utility for managing the lifecycle of virtual machines as a vagrant in that the VMs are not meant to hang around in the same place for long.

Unless you want to pollute your machine with every imaginable programming language, framework and library version you'll find yourself often creating a virtual machine (VM) for each software project. Sometimes more than one. And if you're like me of the past you'll end up with a VirtualBox full of VMs. If you haven't gone about this the right way, you'll end up wondering what VM went with which project and now how did I create it? The anti-pattern around this problem is to write documentation. A better way that aligns with DevOps repeatable practices is to create automation to provision and configure your development VMs. This is where Vagrant comes in as it is "a command line utility for managing the lifecycle of virtual machines."

8.5.1. Documentation and source

Vagrant's documentation can be found at

https://www.vagrantup.com/docs/index.html

It's canonical (i.e., authoritative) source can be found at

https://github.com/hashicorp/vagrant/

Vagrant is written in Ruby. In fact, a Vagrantfile is written in a Ruby DSL.

8.5.2. Installing

1. Download the latest Vagrant release
   https://www.vagrantup.com/downloads.html
   We have verified version 2.0.3 works on Windows
   https://releases.hashicorp.com/vagrant/2.0.3/vagrant_2.0.3_x86_64.msi
   Version 2.0.3 for Mac OS X works fine.
2. Click on the installer once downloaded and follow along. The installer may bury modals you need to respond to in the Windows Task bar, so keep an eye out for that. The installer will automatically add the `vagrant` command to your system path so that it is available on the command line. If it is not found, the documentation advises to try

logging out and logging back into your system. It advises this is particularly necessary sometimes for Windows. Windows will require a reboot, so remember to **come back and complete step-3**.

3. Replace the `vagrant_files/cacert.pem` into `C:\hashicorp\vagrant\embedded` for Windows, and for Mac OS X copy it to `/opt/vagrant/embedded`. On Windows use the File Explorer to replace the existing `cacert.pem` file. The replacement has MITRE's certificates necessary for the SSL inspection hell.

8.5.3. Installing Plugins

In the command line (in `Git Bash`, if on Windows) on the host

```
vagrant plugin install vagrant-proxyconf
vagrant plugin install vagrant-ca-certificates
vagrant plugin install vagrant-cachier
vagrant plugin install vagrant-vbguest
```

Verify with

```
vagrant plugin list
```

Resembles this

```
vagrant-ca-certificates (1.3.0)
vagrant-cachier (1.2.1)
vagrant-proxyconf (1.5.2)
vagrant-vbguest (0.15.1)
```

NOTE

- For lab computers remember to re-authenticate through http://info.mitre.org to be able to access the Internet otherwise `vagrant` will error trying to retrieve the plugins.

8.5.4. The Vagrantfile explained

The Vagrantfile describes how to provision and configure one or more virtual machines.

Vagrant's own documentation puts it best:

Vagrant is meant to run with one Vagrantfile per project, and the Vagrantfile is supposed to be committed to version control. This allows other developers involved in the project to check out the code, run vagrant up, and be on their way. Vagrantfiles are portable across every platform Vagrant supports.

This project's Vagrantfile can be found at the root of the project and it contains the following components I broken apart in order to discuss.

8.5.4.1. Modelines

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
```

When authoring, tells your text editor (e.g. emacs or vim) to choose a specific editing mode for the Vagrantfile. Line one is a modeline for emacs and line two is a modeline for vim.

8.5.4.2. Proxy reset script

```
$proxy_script = <<SCRIPT
tmp=`mktemp`
echo "Remove prior proxy settings across the OS" 2>&1
egrep -vi "^\s*h?[ft]tps?_proxy.*" /etc/environment > $tmp && mv $tmp /etc/environment
egrep -vi "^\s*proxy=.*" /etc/yum.conf > $tmp && mv $tmp /etc/yum.conf
rm -f /etc/profile.d/proxy.sh
[ -e /etc/sysconfig/docker ] && egrep -vi "^\s*h?[ft]tps?_proxy.*" /etc/sysconfig/docker > $tmp && mv $
rm -f $tmp
SCRIPT
```

`$proxy_script` is variable to hold the contents of a script used to remove proxy settings on the OS. I'm working to make it so the class project can be run off the MII.

8.5.4.3. Inserting Proxy setting via host environmental variables

The proxy setting section above uses the `vagrant-proxyconf` plugin to set vagrants (i.e., managed VMs) to use specified proxies by using http_proxy and no_proxy environment variable.

```
Vagrant.configure('2') do |config|

    # Set proxy settings (Depends on vagrant-proxyconf plugin) for all vagrants,
    # if environment holds variables for
    if Vagrant.has_plugin?('vagrant-proxyconf')
      if (ENV['http_proxy'] || ENV['https_proxy'])
        puts "http_proxy = #{ENV['http_proxy']}"
        config.proxy.http = "#{ENV['http_proxy']}"
        puts "https_proxy = #{ENV['https_proxy']}"
        config.proxy.https = "#{ENV['https_proxy']}"
        puts "ftp_proxy = #{ENV['ftp_proxy']}"
        config.proxy.ftp = "#{ENV['https_proxy']}"
        puts "no_proxy = #{ENV['no_proxy']}"
        config.proxy.no_proxy = "#{ENV['no_proxy']}"
        config.proxy.enabled = true
      else
        puts "Not using an http proxy."
        config.proxy.enabled = false
      end
    else
      raise "Missing vagrant-proxyconf plugin.  Install via: vagrant plugin install vagrant-proxyconf"
    end
```

8.5.4.4. Alternatively, inserting hard coded Proxy setting

Or you can hard code the same settings. This later method will likely reduce issues in the class as I found a bug in the `vagrant-proxyconf` plugin.

```
# Hard coded proxy settings
if Vagrant.has_plugin?('vagrant-proxyconf')
  puts "http_proxy = http://gatekeeper.mitre.org:80"
  config.proxy.http = "http://gatekeeper.mitre.org:80" #"http://129.83.31.1:80"
  puts "https_proxy = http://gatekeeper.mitre.org:80"
  config.proxy.https = "http://gatekeeper.mitre.org:80" #"http://129.83.31.1:80"
  puts "ftp_proxy = http://gatekeeper.mitre.org:80"
  config.proxy.ftp = "http://gatekeeper.mitre.org:80" #"http://129.83.31.1:80"
  puts "no_proxy = 127.0.0.1,localhost,.mitre.org,.local,192.168.0.10,192.168.0.11,192.168.0.12"
  config.proxy.no_proxy = "127.0.0.1,localhost,.mitre.org,.local,192.168.0.10,192.168.0.11,192.168.0.
  config.proxy.enabled = true
else
  raise "Missing vagrant-proxyconf plugin.  Install via: vagrant plugin install vagrant-proxyconf"
end
```

8.5.4.5. Inserting MITRE CA Certificates

```
# add MITRE CA Certificates (Depends on vagrant-ca-certificates plugin) to all vagrants
if Vagrant.has_plugin?('vagrant-ca-certificates')
  config.ca_certificates.enabled = true
  config.ca_certificates.certs = [
    "http://pki.mitre.org/MITRE%20BA%20ROOT.crt",
    "http://pki.mitre.org/MITRE%20BA%20NPE%20CA-1.crt",
    "http://pki.mitre.org/MITRE%20BA%20NPE%20CA-3.crt"
  ]
else
  raise "Missing vagrant-ca-certificates plugin.  Install via: vagrant plugin install vagrant-ca-cert
end
```

This section uses `vagrant-ca-certificates` to as the plugin's documentation puts it *configures the virtual machine to inject the specified certificates into the guest's root bundle. This is useful, for example, if your enterprise network has a firewall (or appliance) which utilizes SSL interception.* So, we aren't the only ones that have to deal with the havoc SSL interception brings to development.

8.5.4.6. Configuring the cache plugin to speed things along

```
if Vagrant.has_plugin?("vagrant-cachier")
  # configure cached packages to be shared between instances of the same base box.
  # more info on http://fgrehm.viewdocs.io/vagrant-cachier/usage
  config.cache.scope = :box
else
  raise "Missing vagrant-cachier plugin.  Install via: vagrant plugin install vagrant-cachier"
end
```

This section uses the currently unmainained `vagrant-cachier` plugin to spead up things by sharing common package cache among vagrants.

8.5.4.7. Requiring *vagrant-vbquest* for Windows

```
if Vagrant.has_plugin?('vagrant-vbguest')
else
  raise "Missing vagrant-vbguest plugin.  Install via: vagrant plugin install vagrant-guest"
end
```

`vagrant-vbguest` automatically installs the host's VirtualBox Guest Additions on the guest system. Not needed on Mac OS X, but needed on Windows.

8.5.4.8. Configuring the *dev* vagrant

```
# Provision multiple machines

## provision development vagrant
config.vm.define "dev", primary: true do |dev|
  dev.vm.box = "centos/7"
  dev.vm.network "private_network", ip: "192.168.0.10"
  dev.vm.network :forwarded_port, guest: 22, host: 2222, id: 'ssh'
  dev.vm.hostname = "dev"
  dev.vm.synced_folder ".", "/vagrant", type: "virtualbox"
  dev.vm.provider :virtualbox do |virtualbox|
    virtualbox.name = "DevOps Class - dev"
    virtualbox.customize ["guestproperty", "set", :id, "/VirtualBox/GuestAdd/VBoxService/--timesync-s
    virtualbox.memory = 2048
    virtualbox.cpus = 2
    virtualbox.gui = false
  end

  if !dev.proxy.enabled
    # remove all traces of proxy settings
    dev.vm.provision "shell", run: "always", inline: $proxy_sript
  end

  # configure the vagrant via Ansible on the Vagrant
  config.vm.provision "ansible_local" do |ansible|
    ansible.playbook = "ansible/development-playbook.yml"
    ansible.inventory_path = "hosts"
    ansible.limit = "all"
    ansible.install_mode = "pip"
    ansible.version = "2.4.3.0"
    ansible.compatibility_mode = "2.0"
    ansible.verbose = "vvvv" # true (equivalent to v), vvv, vvvv
  end
end
```

This section provisions and configures a `dev` vagrant used for development

- The `config.vm.provision` block uses `ansible_local` to configure the `dev` vagrant. I've written Ansible roles under `ansible/roles` to automate the configuration of the vagrants.
  - `toolchain.vm.box` loads the vagrant with this `centos/7` vagrant base box. Vagrant curates a listing of base boxes here https://app.vagrantup.com/boxes/search
  - `dev.vm.synced_folder` mounts the class's project folder to `/vagrant` path in the vagrant.
  - The `dev.vm.provider` section tells the hypervisor how to configure the vagrant (i.e., how much memory, how many procesors)

- `ansible_local` does not require one to have Ansible installed on the host. Vagrant handles installing it on the vagrant and executing the specified playbook.
    - In this instance, I have `ansible.verbose` jacked to its highest setting providing loads of *dev* logging. If too much information about what is going on unnerves you changing this to `false` will disable verbose logging.

8.5.4.9. Configuring the *toolchain* vagrant

```
## provision the pipeline vagrant
config.vm.define "toolchain", autostart: false do |toolchain|
  toolchain.vm.box = "centos/7"
  toolchain.vm.network "private_network", ip: "192.168.0.11"
  toolchain.vm.network :forwarded_port, guest: 22, host: 2223, id: 'ssh'
  toolchain.vm.hostname = "toolchain"
  toolchain.vm.synced_folder ".", "/vagrant", type: "virtualbox", mount_options: ['dmode=777','fmode=
  toolchain.vm.provider :virtualbox do |virtualbox|
    virtualbox.name = "DevOps Class - toolchain"
    virtualbox.customize ["guestproperty", "set", :id, "/VirtualBox/GuestAdd/VBoxService/--timesync-s
    virtualbox.customize ["modifyvm", :id, "--clipboard", "bidirectional"]
    virtualbox.customize ["modifyvm", :id, "--draganddrop", "bidirectional"]
    virtualbox.memory = 4096
    virtualbox.cpus = 4
    virtualbox.gui = false
  end

  if !toolchain.proxy.enabled
    # remove all traces of proxy settings
    toolchain.vm.provision "shell", run: "always", inline: $proxy_sript
  end

  # configure the vagrant via Ansible on the Vagrant
  config.vm.provision "ansible_local" do |ansible|
    ansible.playbook = "ansible/toolchain-playbook.yml"
    ansible.inventory_path = "hosts"
    ansible.limit = "all"
    ansible.install_mode = "pip"
    ansible.version = "2.4.2.0"
    ansible.compatibility_mode = "2.0"
    ansible.verbose = "vvvv" # true (equivalent to v), vvv, vvvv
  end
end
```

This section provisions and configures the `toolchain` vagrant. This is the beefy vagrant running GitLab. Drone CI, the private Docker registry, SonarQube, Selenium, Taiga...

8.5.4.10. Configuring the vagrants from one playbook

```
  # # configure the vagrants via Ansible on the Vagrant
  # config.vm.provision "ansible_local" do |ansible|
  #   ansible.playbook = "ansible/playbook.yml"
  #   ansible.inventory_path = "hosts"
  #   ansible.limit = "all"
  #   ansible.install_mode = "pip"
  #   ansible.version = "2.4.2.0"
  #   ansible.compatibility_mode = "2.0"
  #   ansible.verbose = true # true (equivalent to v), vvv, vvvv
  # end
end
```

Instead of individually configuring each VM one could comment out vagrants individual `config.vm.provision` and uncomment this section to configure all the vagrants from one single playbook.

8.6. Ansible

Ansible is a "configuration management" tool that automates software provisioning, configuration management, and application deployment. Configuration management, deployment automation are two core repeated practices in DevOps, so for the purposes of the class Ansible addresses this concern in configuring the two vagrants.

Ansible was open source and then subsumed by Red Hat.

There are other "configuration management" tools, such as Chef and Puppet. There are of course even more, but they hold little or no market share (e.g., BOSH, Salt.)

8.6.1. Playbooks

In Ansible one defines playbooks to manage configurations of and deployments to remote machines. The playbooks I'm using to configure the vagrants exist in `ansible/` .

The `toolchain` vagrant Ansible playbook (`ansible/development-playbook.yml`) contains:

```
---
# Ansible playbook for toolchain

# Copyright (C) 2018 Michael Joseph Walsh - All Rights Reserved
# You may use, distribute and modify this code under the
# terms of the the license.
#
# You should have received a copy of the license with
# this file. If not, please email <mjwalsh@nemonik.com>

- hosts: toolchains
  remote_user: vagrant
  roles:
    - common
    - docker
    - docker-compose
    - docker-registry
    - taiga
    - gitlab
    - drone
    - sonarqube
    - golang-container-image
    - python-container-image
    - golang-sonarqube-scanner-image
    - standalone-firefox-container-image
    - owasp-zap2docker-stable-image
#    - standalone-chrome-container-image
#    - gnome-desktop
#    - firefox
#    - chrome
  vars:
    docker_compose_version: "1.21.0"
```

The file is written in a YAML-based DSL (domain specific language.)

8.6.2. Roles

The roles exist in `ansible/roles` and permit Roles easy sharing of bits of configuration content with other users. Roles can also be found in the Ansible Galaxy, retrieved and placed into the `ansible/roles` folder to be used, but I wrote all the roles for the class.

A role, such as a Taiga role is comprised of many components (e.g., files, templates), but at its core is the main task. The main task for the `taiga` roles contains

```
---
# tasks file for taiga

# Copyright (C) 2018 Michael Joseph Walsh - All Rights Reserved
# You may use, distribute and modify this code under the
# terms of the the license.
#
# You should have received a copy of the license with
# this file. If not, please email <mjwalsh@nemonik.com>

- name: copy taiga files to /root/taiga
  become: yes
  copy:
    src: files/
    dest: /root/taiga

- name: make /root/taiga/dockerfile_build.sh executable
  become: yes
  file:
    path: /root/taiga/dockerfile_build.sh
    mode: "u=rwx,g=r,o=r"

- name: template files into /root/taiga
  become: yes
  template:
    src: "{{ item.src }}"
    dest: "{{ item.dest }}"
  with_items:
    - { src: "templates/README.j2", dest: "/root/taiga/README.MD" }
    - { src: "templates/docker-compose.j2", dest: "/root/taiga/docker-compose.yml" }
    - { src: "templates/taiga-front-dist/dist/conf.j2", dest: "/root/taiga/volumes/taiga-front-dist/dis

- name: build nemonik/taiga:latest image w/ buildargs to handle possible http proxy
  become: yes
  docker_image:
    path: /root/taiga
    name: nemonik/taiga:latest
    buildargs:
      https_proxy: "{{ ansible_env.HTTP_PROXY }}"
      http_proxy: "{{ ansible_env.HTTP_PROXY }}"
      HTTP_PROXY: "{{ ansible_env.HTTP_PROXY }}"
      HTTPS_PROXY: "{{ ansible_env.HTTP_PROXY }}"
      NO_PROXY: "{{ ansible_env.NO_PROXY }}"
      no_proxy: "{{ ansible_env.NO_PROXY }}"

- name: spin up taiga via docker-compose
  become: yes
  docker_service:
    build: no
    debug: yes
    project_src: /root/taiga/
```

The `Taiga` role dependes on `docker-compose`, `docker`, and `common` having configured the vagrant.

8.6.2.1. Docker image and containers

So, what is Docker? What are container images? What are Containers?

A container image is a lightweight, self-contained, executable package that includes everything needed to run your application including runtime, system tools, system libraries, and settings.

A container is essentially an isolated processes running in user space. The benefit over Virtual Machines in that multiple containers can run on the same machine (in case of this class, a vagrant) sharing the OS kernel with other containers. Whereas a VM would require a full copy of an operating system in order to run the application. This makes containers insanely light-weight.

More can be read on the topic at

https://www.docker.com/what-container

You will build a Docker image and spin up a container in this class.

8.6.2.2. Docker image and containers

And what is docker-compose?

Doceker-compose is a tool and domain specific language based on YAML used to define and run multi-container Docker applications.

What is YAML? YAML bills itself as *a human-friendly data serialization standard for all programming languages.* YAML also follows in in the computing tradition of being a recursive acronym, *YAML Ain't Markup Language.* Many of the tools used in this course make use of YAML, so you will see plenty examples of it.

You will edit a `docker-compose.yml` file in this class in order to complete the configuration of one the key CI/CD tools.

8.7. Spinning up the *toolchain* vagrant

In the command line of the host in the root of the class project, open `ansible/tool-chain-playbook.yml` and make sure the roles are as so:

```
---
# Ansible playbook for toolchain

# Copyright (C) 2018 Michael Joseph Walsh - All Rights Reserved
# You may use, distribute and modify this code under the
# terms of the the license.
#
# You should have received a copy of the license with
# this file. If not, please email <mjwalsh@nemonik.com>

- hosts: toolchains
  remote_user: vagrant
  roles:
    - common
    - docker
    - docker-compose
    - docker-registry
    - taiga
    - gitlab
    - drone
    - sonarqube
    - golang-container-image
    - python-container-image
    - golang-sonarqube-scanner-image
    - standalone-firefox-container-image
    - owasp-zap2docker-stable-image
#    - standalone-chrome-container-image
#    - gnome-desktop
#    - firefox
#    - chrome
  vars:
    docker_compose_version: "1.21.0"
```

Then enter into the command-line of the host at the root of the class project

```
vagrant up toolchain
```

You will see a good deal of output and on the Windows OS it will pester you to approve certain things. Blindly approve everything.

Once complete open a secure shell (ssh) to the `toolchain` vagrant

```
vagrant ssh toolchain
```

The command line will open a prompt a bash shell on the vagrant

```
[vagrant@toolchain ~]$
```

On this vagrant will be running a number of DevOps tools. This will take a while, so let's discuss what is being installed. Caution, it is easy to forget DevOps is as much about culture as it is about a methodology and repeated practices (often further mistakenly thought as "tools and automation"), so keep this in mind. The tools, methodology, an repeated practices exist to support the culture.

8.7.1. Taiga, an example of Agile project management software

Taiga is an Open Source project management platform for agile development.

There are many project management platforms for Agile.

Typically, Agile teams work using a visual task management tool such as a project board, task board or Kanban or Scrum visual management board. These boards can be implemented using a whiteboard or open space on a wall or in software. The board is at a minimum segmented into a few columns *To do*, *In process*, and *Done*, but the board can be tailored. I've personally seen boards for very large projects consume every bit of wallspace of a very large cavernous room, but as Lean-Agile has matured, teams have grown larger and more disparate, tools have emerged to provide a clear view into a project's management to all levels of concern (e.g., developers, managers, product owner, and the customer) answering:

- Are deadlines being achieved?
- Are team members overloaded?
- How much is complete?
- What's next?

Further, the Lean-Agile Software tools should provide the following capabilities:

- Dividing integration and development effort into multiple projects.
- Defining, allocating, and viewing resources and their workload across each product.
- Defining, maintaining, and prioritizing the accumulation of an individual product's requirements, features or technical tasks which, at a given moment, are known to be necessary and sufficient to complete a project's release.
- Facilitating the selection and assignment of individual requirements to resources, and the tracking of progress for a release.
- Permit collaboration with external third parties.

The 800lb Gorilla in this market segment is JIRA Software. Some of my co-workers hate it. It is part of the Atlassian suite providing provides collaboration software for teams with products including JIRA Software, Confluence, HipChat, Bitbucket, and Stash.

MITRE provides:

- JIRA Software as part of CODEV at https://tracker.codev.mitre.org/secure/Dashboard.jspa where you have to first login via https://login.codev.mitre.org/portal.
- Taiga at https://taiga.mitre.org/ using your MITRE username and password to log in.

There may be more. I'm simply not aware of them.

A number of MITRE projects also use Pivotal Lab's Pivotal Tracker, a SaaS.

8.7.1.1. Documentation, source, container image

Taiga's documentation can be found at

It's canonical source can be found at

https://github.com/taigaio/taiga-front-dist/

dedicated to the front-end, and

https://github.com/taigaio/taiga-back/

dedicated to the back-end.

Taiga doesn't directly offer a Docker container image for, but I've authored an image that collapses both taiga-front-dist and -back onto one container behind an NGINX reverse proxy.

8.7.1.2. URL, Username and password

Once, stood up your instance of Taiga will reachable at

http://192.168.0.11:8080/

The default admin accont username and password are

admin 123123

8.7.2. GitLab CE, an example of configuration management software

GitLab is installed on the `toolchain` vagrant, where it will be accessible at http://192.168.0.11:10080/.

GitLab Community Edition (CE) is an open source end-to-end software development platform with built-in version control, issue tracking, code review, and CI/CD.

For Agile teams to collaborate, a Configuration management (CM) is necessary to coordinate the development of new feature, changes, and experimentation. Also, a CM system (CMS) provides a history of changes, and thereby, the ability to roll back to a version known to be acceptable.

At a minimum, the following items will be placed under revision control in CM:

- Source code,
- If a database is needed, schema initialization and the migration between versions,
- Text documentation containing
  - a synopsis (i.e., project name, overview, etc.),
  - version description,
  - guidance covering
    - build,
    - unit testing, and
    - installation
  - a contributor enumeration,
  - license and/or ownership declaration with contacts, etc.,

A single CMS and the associated workflow (e.g., GitHub Workflow) can serve as the focal point for the entire enterprise thereby provide centralized version control if all documentation is authored in a lightweight markup language with plain text formatting syntax (e.g., Markdown, PlantUML).

A CMS must facilitate best practices, not limited to:

- A means for developers to copy and work off a complete repository thereby permitting
  - Private individual work to later be synchronized via exchanging sets of changes (i.e., patches) through a means described as "distributed version control", and
  - Pre-flight build and test of their source code in their own private workspace, so as to minimize the chance of committing broken or untested source code thereby encouraging
  - The committing of completed source code only.
- Granular commits that communicate the motivation for the commit (i.e., the what and why). For example, for a change these could be:
  - the inclusion of a new feature,
  - a bug fix,
  - the removal of dead code
- Reducing the risk breaking a build by
  - Utilize branching to separate different lines of development, and
  - Standardize on CMS workflows (e.g., GitHub Workflow),
- Make builds be self-testing (i.e., ingrain testing) by including unit and integration test with the source code so that it can be executed by
  - the build automation, and
  - the Continuous Integration service.
- Trigger follow-on activities orchestrated by the Continuous Integration Service.

MITRE provides:

- A GitLab at https://gitlab.mitre.org/users/sign_in uses your MITRE username and password to authenticate. This is where you retrieved this class, so you already knew this.
- Bitbucket (formally Stash) under CODEV at https://git.codev.mitre.org/dashboard where you have to first login via https://login.codev.mitre.org/portal.

8.7.2.1. Documentation, source, container image

GitLab's documentation can be found at

https://docs.gitlab.com/ce/

It's canonical source can be found at

https://hub.docker.com/r/sameersbn/gitlab/

https://gitlab.com/gitlab-org/gitlab-ce

I'm using Sameer Naik's Docker container image for GitLab. The image can be found at

https://hub.docker.com/r/sameersbn/gitlab/

And its canonical source is located at

https://github.com/sameersbn/docker-gitlab

8.7.2.2. URL, Username and password

Once, stood up your instance of GitLab will reachable at

http://192.168.0.11:10080/

GitLab requires you to enter a password for its **root** account. You will be using the root account to host your repositories vice creating your own, but if you want you can. There is nothing stopping you.

I would suggest for the purposes of the class choosing something simple, but at least the same number of characters as "password".

8.7.3. Drone CI, an example of CI/CD orchestrator

Drone CI often referred to simply as "Drone" is installed on the `toolchain` vagrant, where it will be accessible at http://192.168.0.11:8000/ after some user configuration that will be explained later in the *Integrate Drone CI with GitLab* section.

Drone is essentially a Continuous Delivery system built on container technology.

Drone is distributed as a Docker image. Drone CI can be run with an internal SQLite database, but it is advisable to run with an external database and this the configuration the class uses. It also integrates with multiple version control providers (i.e., GitHub, GitLab, BitBucket, Stash, and Gogs). Both CMS and database are configured using environment variables passed when the Drone CI container is first to run. Drone plugins (really just containers) can be used to deploy code, publish artifacts, send a notification, etc. Drone's approach to plugins is novel as plugins are really just Docker containers distributed in a typical manner. Each plugin is designed to perform pre-defined tasks and is configured as steps in your pipeline. Plugins are executed with read/write/execution access at the root of the source branch, therefore, permitting the pipeline to interact with the specific branch of source to build, test, bundle, deliver, and deploy. Developers create a Drone CI pipeline by placing a `.drone.yml` file in the root of the repository. The .drone.yml is authored in a domain specific language (DSL) that is a superset of the docker-compose DSL to describe the build with multiple named steps executed in a separate Docker container having shared disk access to the specific branch of the source repository.

Drone and its brethren (e.g., Jenkins CI, GitLab CI/CD) is used to facilitate Continuous Integration (CI), a software development practice where members of an Agile team frequently integrate their work in order to detect integration issues as soon as possible. Each integration is orchestrated through a service that essentially assembles a build and runs tests every time a predetermined trigger has been met; and then reports with immediate feedback.

MITRE provides a Jenkins CI server at https://jenkins.mitre.org/. Use you MITRE username and password to authenticate and you can have at all that Jenkins CI goodness. I don't use it unless I have to. Why? I'm simply not a fan. Initially, because its plugin architecture is painful to manage and your pipelines existed entirely in the Jenkins CI tool itself. Later, Jenkins CI introduced Groovy-based Jenkin Pipelines that were CMed wit your project's source. Every orchestrator has based their DSL on YAML and although I love the Groovy language for its power, I don't find it makes for a good orchestration language. Your opinion may differ. I'm okay with that.

There are also SaaS CI/CD tools, such as Travis CI and Circle CI. These are great, free CI/CD orchestrators.

My java-stix project hosted on GitHub.com uses both Travis CI and Circle CI as part of its continuous integration.

The Travis CI orchestration contains

```
language:  java
dist: precise
jdk:
  - oraclejdk7

before_install:
  - chmod +x gradlew

env: GRADLE_OPTS=-Dorg.gradle.daemon=true
env: CI_OPTS=--stacktrace

install: /bin/true
script: "./gradlew -x signArchives"
```

Whereas, the Circle CI orchestration contains

```
machine:
  java:
    version: oraclejdk7
  environment:
    GRADLE_OPTS: -Dorg.gradle.daemon=true
    CI_OPTS: --stacktrace --debug

test:
  override:
    - ./gradlew -x signArchives
```

They are essentially similar both requiring the Oracle JDK and use Gradle to build and unit test the code. Both SaaS under the covers uses containers to run the builds.

In another GitHub hosted project, the java-stix-validator the Travis CI orchestration contains

```
language: java
jdk:
- oraclejdk8
before_install:
- chmod +x gradlew
env: CI_OPTS=--stacktrace
install: "/bin/true"
script: "./gradlew build -d"
deploy:
  provider: heroku
  api_key:
    secure: m9Gbt0Oyqtjwyu4Y8CVobNNnj1q5mFt+Ygi2wiDWlf/RunLOj2CE8YAYuRyEAbpCOd1lrmrhmQb8uQAfiydauYBcQE5
  app: agile-journey-9583
  on:
    repo: STIXProject/java-stix-validator
```

To deploy the web application to Heroku, one of the first PaaS (Platforms-as-Service). The code was last committed in 2015 and the code is still running free on Heroku at http://agile-journey-9583.herokuapp.com/#/

More details on Drone is sprinkled across the class. As you can see I favor being a polyglot when it comes to CI/CD.

8.7.3.1. Documentation, source, container image

Drone's main site is at

https://drone.io/

Its documentation is at

http://docs.drone.io/

Its plugin market is at

http://plugins.drone.io/

Drone's canonical source can be found at

https://github.com/drone/drone

Drone is distributed as a container image and can be found respectfully at

https://hub.docker.com/r/drone/drone/

and

https://hub.docker.com/r/drone/agent/

8.7.3.2. URL, Username and password

Once, stood up your instance of Drone CI will reachable at

http://192.168.0.11:8000/

Drone will authenticate you off GitLab once integrated.

8.7.4. Integrate Drone CI with GitLab

Once, Drone is configured, accomplish the steps.

The Ansible automation will install patched docker containers for Drone and template the docker-compose file used to spin it up, but the automation will not spin up Drone as it requires human intervention in the absence of automation.

So, you will need to accomplish the following steps, while at the same time becoming familiar with GitLab.

1. As `Root` log into GitLab with the password you entered in the prior step,
2. Click on the `wrench` to enter the `Admin Area`,
3. Click `Applications`,
4. Click `New Application`,
5. Enter `Drone CI` for `Name`,
6. Enter `http://192.168.0.11:8000/authorize`,
7. Click `submit`,
8. In a command line on the host (i.e., the machine running the vagrants) at the root of the class project enter into the host's command line at the root of the project:

```
vagrant ssh toolchain
```

9. On the `toolchain` vagrant enter into the command line

```
sudo su
cd /root/drone
```

10. Around line 50, apply the `Application Id` and `Secret` provided by GitLab by setting `DRONE_GITLAB_CLIENT` and `DRONE_GITLAB_SECRET` respectively by editing `docker-compose.yml` .

```
      - DRONE_GITLAB_CLIENT=Your Application Id
      - DRONE_GITLAB_SECRET=Your Secret
```

Delete `Your Application Id` and `Your Secret` text and replace with `Application Id` and `Secret` , respectively.

11. Then enter into the command line

```
docker-compose up -d
```

Command line output will resemble

```
[vagrant@toolchain drone]$ ./start.sh
Creating drone_drone-server_1     ... done
Creating drone_drone-server_1     ...
Creating drone_drone-agent_1      ... done
```

You can drop `-d` on the `docker-compose` so as to not daemonize the drone, so you can monitor its logging for issues.

12. Exit the root session.

```
exit
```

13. Head over or http://192.168.0.11:8000 and click `Authorize` to permit Drone to use your GitLab account.

You will only have to do these step in regards to Drone once. Drone should always restart in the toolchain vagrant even if you `halt` the vagrant and later bring it `up` .

8.7.5. SonarQube, an example of a platform for the inspection of code quality

SonarQube provides the capability to show the health of an application's source code, highlighting issues as they are introduced. SonarQube can be extended by language-specific extensions/plugins to report on duplicated code, coding standards, unit tests, code coverage, code complexity, comments, bugs, and security vulnerabilities.

8.7.5.1. Documentation, source, container image

SonarQube's main site is at

https://www.sonarqube.org/

Its documentation is at

https://docs.sonarqube.org/display/SONAR/Documentation

SonarQube's canonical source can be found here

https://github.com/SonarSource/sonarqube

I'm using the container image provided at

https://hub.docker.com/_/sonarqube/

8.7.5.2. URL, Username and password

Once, stood up your instance of SonarQube will reachable at

http://192.168.0.11:9000/

The default admin account username and password is

admin

8.8. Spin up the *dev* vagrant

In another command line of the host in the root of the class project

Open `ansible/tool-chain-playbook.yml` and make sure these roles are uncommented:

```
---
# Ansible playbook for dev vagrant

# Copyright (C) 2018 Michael Joseph Walsh - All Rights Reserved
# You may use, distribute and modify this code under the
# terms of the the license.
#
# You should have received a copy of the license with
# this file. If not, please email <mjwalsh@nemonik.com>

- hosts: [development]
  remote_user: vagrant
  roles:
    - common
    - golang
    - golint
    - docker
    - drone-cli
    - docker-compose
    - sonar-scanner
  vars:
    golang_version: "1.10.1"
    docker_compose_version: "1.21.0"
```

Then in the command-line of the host (not while ssh'ed into the toolchain vagrant) at the root of the class project

```
vagrant up dev
```

You will see a good deal of output.

Once complete open a secure shell to the `dev` vagrant

```
vagrant ssh dev
```

Command line will open a prompt to the vagrant

```
[vagrant@dev ~]$
```

8.9. Golang *helloworld* project

The prior `toolchain` and `dev` vagrants are required to be up and running for the following sections.

In this next part, we will create a simple helloworld GoLang project to demonstrate Continuous Integration. GoLang lends itself well to DevOps and underlines almost every new tool you can think of related to DevOps and cloud (e.g., golang / go, docker / docker-ce, kubernetes / kubernetes, hashicorp / terraform, coreos / etcd, hashicorp / vault, hashicorp / packer, hashicorp / consul, gogits / gogs, drone / drone.)

8.9.1. Create the project's backlog

A backlog is essentially your to-do list, a prioritized list of work derived from the roadmap (e.g., the outline for future product functionality and when new features will be released) and its requirements.

Open Taiga in your web browser

http://192.168.0.11:8080

The default admin account username and password are

admin 123123

Complete the follow to track your progress in completing the Golang *helloworld* project

1. Click `Create Project` .
2. Select `Kanban` . A Kanban board shows how work moves from left to right, each column represents a stage within the value stream.
3. Give your project a name. For example, `Helloworld` and a description, such as, `My Kanban board for this awesome helloworld app` and then click `CREATE PROJECT` .
4. You can skip this step and opt to chose to click `><` to fold `READY` , `USER STORY STATUS` and `ARCHIVED` only after completing step 6. Otherwise, you can edit your Kanban board to just `NEW` , `IN PROGRESS` , and `DONE` by
   a. On the bottom left, click the `ADMIN` gear.
   b. Click `ATTRIBUTES` .

c. Scroll down to `USER STORY STATUS`.

d. Hover over `Ready`, click the trash icon to delete and click `ACCEPT`.

c. Do the same for `Ready for test` and `Archived`.

d. Click the `KANBAN` icon on the far left. It Looks like columns. And then reload the browser to get the changes to take

5. In the `NEW` column select `Add New bulk` icon that looks like a list and when the page updates cut-and-paste the lines below into the text box and click `SAVE`.

```
Create the project's backlog
Create the project in GitLab
Setup your project on the dev Vagrant
Author the application
Align source code with Go coding standards
Lint your code
Build the application
Execute your application
Author a unit test
Run sonar-scanner on the application
Register your app in SonarQube
Run the sonar-scanner on the command line
Write build and test automation
Author Drone-based Continuous Integration
Configure Drone to execute your pipeline
Trigger the build
```

Track your progress in Taiga as you work through each section.

8.9.2. Create the project in GitLab

1. In GitLab (http://192.168.0.11:10080/) click on `Projects` in the upper left.
   a. Select `Create Project`.
   b. Or click http://192.168.0.11:10080/projects/new
2. Leave the `Project path` defaulted to `http://192.168.0.11:10080/root/`.
3. Enter `helloworld` (the form field the page defaults to) for the `Project name`.
4. Provide an optional `Project description`. Something descriptive, such as, "*GoLang helloworld application for the hands-on DevOps class.*".
5. Make the application `public` to save yourself from entering your username and password when cloning.
6. Click the green `Create Project` button on the lower left.

The UI will refresh to show you a landing page for the project that should be accessible from http://192.168.0.11:10080/root/helloworld

8.9.3. Setup your project on the *dev* Vagrant

On your host open in a shell to `dev` vagrant configure your user name and email:

```
git config --global user.name "Administrator"
git config --global user.email "admin@example.com"
```

The Ansible `golang` role (Found at `ansible/roles/golang` in the class project on your host.) will already have configured:

1. Your `$GOPATH` and `$GOBIN` environmental variable.
2. Added `$GOBIN` to your PATH environmental variable.
3. Created `go` workspace in vagrant user's home directory (i.e., `/home/vagrant/go` directory containing `bin`, `pkg`, and `src`.)

Source code for Go code is placed in the `src` directory under a namespace (i.e., a unique base path to avoid naming collisions under which all your go code will reside.) In open source software development, it is typical to use github.com account path. Mine is `github.com/nemonik`, so I would create this base path via (You do the same., so we are all on the same page.)

```
mkdir $GOPATH/src/github.com/nemonik
```

Now lets create the GoLang `helloworld` project to demonstrate Continuous Integration via

```
cd $GOPATH/src/github.com/nemonik
git clone http://192.168.0.11:10080/root/helloworld.git
cd helloworld
```

So that you do not commit certian files to GitLab when you push, create a `.gitignore` file with your editor with this content

```
# OS-specific
.DS_Store

.scannerwork

helloworld
```

8.9.4. Author the application

In the project follder (i.e., `/home/vagrant/go/src/github.com/nemonik/helloworld`), create `main.go` in emacs, nano, vi, or vim with this content:

```
package main

import "fmt"

func main() {
  fmt.Println(HelloWorld())
}

func HelloWorld() string {
  return "hello world"
}
```

8.9.5. Align source code with Go coding standards

Format source code according to Go coding standards using

```
go fmt
```

Results in the code being formatted to

```
package main

import "fmt"

func main() {
    fmt.Println(HelloWorld())
}

func HelloWorld() string {
    return "hello world"
}
```

You'll see the difference if you `cat` your source

```
cat main.go
```

8.9.6. Lint your code

Already installed on your `dev` vagrant is `golint`. Where `go fmt` reformatted the code to GoLang standards, `golint` prints style mistakes.

To run `golint`, in the root of the `helloworld` project execute

```
golint
```

Command line out will be

```
[vagrant@dev helloworld]$ golint
hello.go:9:1: exported function HelloWorld should have comment or be unexported
```

Fix the error by editing `main.go` to

```
package main

import "fmt"

func main() {
  fmt.Println(HelloWorld())
}

// HelloWorld returns "hello world"
func HelloWorld() string {
  return "hello world"
}
```

Run `golint` again and it should return no output indicating it sees nothing wrong.

8.9.7. Build the application

Build the project by executing

```
go build –o helloworld .
```

Success returns no command line output. What? Did you want a cookie? No cookie for you. This is GoLangs way of doing things. Silence is golden and means things went fine. Otherwise, go back and fix the mistakes in your code.

8.9.8. Execute your application

Execute your application

```
./helloworld
```

The command line output will be

```
[vagrant@dev hello]$ ./helloworld
hello world
```

8.9.9. Author a unit test

GoLang ships with a built-in `testing` package

https://golang.org/pkg/testing/

for automated unit testing of Go packages. Unit testing is a software development process where the smallest testable components of an application are individually tested for proper operation. Unit testing offers the biggest return for dollars spent in comparison to integration and functional testing.

For more on this topic read Martin Fowler's

https://martinfowler.com/bliki/TestPyramid.html

In your editor create `main_test.go` with this content:

```
package main

import (
  "os"
  "testing"
)

func TestMain(m *testing.M) {
  os.Exit(m.Run())
}

func TestHelloWorld(t *testing.T) {
  if HelloWorld() != "hello world" {
    t.Errorf("got %s expected %s", HelloWorld(), "hello world")
  }
}
```

Execute the unit test by entering

```
go test
```

The command line returns

```
[vagrant@dev helloworld]$ go test
PASS
ok      github.com/nemonik/helloworld    0.003s
```

This step and all the proceeding follows of a DevOps tenant of Devops, "Developers are expected to pre-flight new code."

8.9.10. Run sonar-scanner on the application

SonarQube provides the capability to show the health of an application's source code, highlighting issues as they are introduced.

8.9.10.1. Register your app in SonarQube

These section is optional as the `sonar-scanner` command line tool will do this for you.

Open in your browser

http://192.168.0.11:9000/admin/projects_management

If you have to authenticate, your username and password is

admin

Click `Create project`.

Provide the `name` of

helloworld

Provide the `key` of

helloworld

8.9.10.2. Run the *sonar-scanner* on the command line

In the command line

```
go get github.com/alecthomas/gometalinter
go get github.com/axw/gocov/...
go get github.com/AlekSi/gocov-xml
go get github.com/jstemmer/go-junit-report
gometalinter --install
gometalinter --checkstyle > report.xml
gocov test ./... | gocov-xml > coverage.xml
go test -v ./... | go-junit-report > test.xml
sonar-scanner -D sonar.host.url=http://192.168.0.11:9000 -D sonar.projectKey=helloworld -D sonar-scanner -D sonar.project
```

Let me unpack what the above commands are doing

- The first 4 lines starting with `go get` install dependencies for linting, coverage reporting, report conversioning.
- The lines ending with an XML file being created (e.g., `report.xml`, `coverage.xml` and `test.xml`) are the reports that will be submitted to SonarQube.
- `sonar-scanner` through `-D` parameters is configured to submit the reports. If you skipped the prior section, SonarQube will automatically create the project for you.

Output written to the command line will resemble

```
[vagrant@dev helloworld]$ go get github.com/alecthomas/gometalinter
[vagrant@dev helloworld]$ go get github.com/axw/gocov/...
[vagrant@dev helloworld]$ go get github.com/AlekSi/gocov-xml
[vagrant@dev helloworld]$ go get github.com/jstemmer/go-junit-report
[vagrant@dev helloworld]$ gometalinter --install
Installing:
  deadcode
  dupl
  errcheck
  gas
  goconst
  gocyclo
  goimports
  golint
  gosimple
  gotype
  gotypex
  ineffassign
  interfacer
  lll
  maligned
  megacheck
  misspell
  nakedret
  safesql
  staticcheck
  structcheck
  unconvert
  unparam
  unused
  varcheck
  vet
[vagrant@dev helloworld]$ gometalinter --checkstyle > report.xml
[vagrant@dev helloworld]$ gocov test ./... | gocov-xml > coverage.xml
ok      github.com/nemonik/helloworld   0.003s  coverage: 50.0% of statements
[vagrant@dev helloworld]$ go test -v ./... | go-junit-report > test.xml
[vagrant@dev helloworld]$ sonar-scanner -D sonar.host.url=http://192.168.0.11:9000 -D sonar.projectKey=helloworld -D sona
INFO: Scanner configuration file: /usr/local/sonar-scanner-3.1.0.1141-linux/conf/sonar-scanner.properties
INFO: Project root configuration file: NONE
INFO: SonarQube Scanner 3.1.0.1141
INFO: Java 1.8.0_121 Oracle Corporation (64-bit)
INFO: Linux 3.10.0-693.21.1.el7.x86_64 amd64
INFO: User cache: /home/vagrant/.sonar/cache
INFO: SonarQube server 7.0.0
INFO: Default locale: "en_US", source code encoding: "UTF-8" (analysis is platform dependent)
INFO: Publish mode
INFO: Load global settings
INFO: Load global settings (done) | time=327ms
INFO: Server id: AWLZ3KmeIXzDnCPHcQJq
INFO: User cache: /home/vagrant/.sonar/cache
INFO: Load plugins index
INFO: Load plugins index (done) | time=97ms
INFO: Load/download plugins
INFO: Load/download plugins (done) | time=216ms
INFO: Process project properties
INFO: Load project repositories
INFO: Load project repositories (done) | time=84ms
INFO: Load quality profiles
```

```
INFO: Load quality profiles (done) | time=218ms
INFO: Load active rules
INFO: Load active rules (done) | time=1580ms
INFO: Load metrics repository
INFO: Load metrics repository (done) | time=135ms
WARN: SCM provider autodetection failed. No SCM provider claims to support this project. Please use sonar.scm.provider to
INFO: Project key: helloworld
INFO: -------------  Scan helloworld
INFO: Load server rules
INFO: Load server rules (done) | time=45ms
INFO: Base dir: /home/vagrant/go/src/github.com/nemonik/helloworld
INFO: Working dir: /home/vagrant/go/src/github.com/nemonik/helloworld/.scannerwork
INFO: Source paths: .
INFO: Source encoding: UTF-8, default locale: en_US
INFO: Index files
INFO: Excluded sources:
INFO:   **/*test.go
INFO: 5 files indexed
INFO: 1 file ignored because of inclusion/exclusion patterns
INFO: Quality profile for go: Golint Rules
INFO: Sensor GoMetaLinter issues loader sensor [golang]
INFO: Parsing the file report.xml
INFO: Parsing 'GoMetaLinter' Analysis Results
INFO: Sensor GoMetaLinter issues loader sensor [golang] (done) | time=37ms
INFO: Sensor Go Coverage [golang]
INFO: /home/vagrant/go/src/github.com/nemonik/helloworld
INFO: Analyse for /home/vagrant/go/src/github.com/nemonik/helloworld/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/refs/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/refs/heads/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/refs/tags/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/branches/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/hooks/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/info/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/objects/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/objects/pack/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/objects/info/coverage.xml
INFO: Sensor Go Coverage [golang] (done) | time=47ms
INFO: Sensor Go test JUnit loader sensor [golang]
INFO: base dir /home/vagrant/go/src/github.com/nemonik/helloworld
WARN: The key is null
INFO: Sensor Go test JUnit loader sensor [golang] (done) | time=6ms
INFO: Sensor Go Highlighter Sensor [golang]
INFO: Sensor Go Highlighter Sensor [golang] (done) | time=27ms
INFO: Sensor Go Metrics Sensor [golang]
INFO: Sensor Go Metrics Sensor [golang] (done) | time=1ms
INFO: Sensor Zero Coverage Sensor
INFO: Sensor Zero Coverage Sensor (done) | time=12ms
INFO: Sensor CPD Block Indexer
INFO: Sensor CPD Block Indexer (done) | time=0ms
INFO: No SCM system was detected. You can use the 'sonar.scm.provider' property to explicitly specify it.
INFO: Calculating CPD for 0 files
INFO: CPD calculation finished
INFO: Analysis report generated in 58ms, dir size=3 KB
INFO: Analysis reports compressed in 8ms, zip size=2 KB
INFO: Analysis report uploaded in 1422ms
INFO: ANALYSIS SUCCESSFUL, you can browse http://192.168.0.11:9000/dashboard/index/helloworld
INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted analysis rep
INFO: More about the report processing at http://192.168.0.11:9000/api/ce/task?id=AWLaCkiXE0IcvTpxuSns
INFO: Task total time: 4.257 s
```

```
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
INFO: Total time: 7.262s
INFO: Final Memory: 7M/78M
INFO: ------------------------------------------------------------------------
```

Then open in your host's web browser

http://192.168.0.11:9000/dashboard?id=helloworld

to view your SonarQube report.

We have some code coverage issues, but otherwise, things look fine.

8.9.11. Write build and test automation

Build automation is a key practice of CI. So, let's make the build reproducible be automate everything we've done this far via authoring a Makefile.

In the root of the project edit the file, `Makefile` and add the following contents

```
GOCMD=go
GOFMT=$(GOCMD) fmt
GOBUILD=$(GOCMD) build
GOCLEAN=$(GOCMD) clean
GOTEST=$(GOCMD) test
GOGET=$(GOCMD) get
BINARY_NAME=helloworld

SONAR_DEPENDENCIES := \
    github.com/alecthomas/gometalinter \
    github.com/axw/gocov/... \
    github.com/AlekSi/gocov-xml \
    github.com/jstemmer/go-junit-report

all: fmt lint test build
clean:
    $(GOCLEAN)
    rm -f $(BINARY_NAME)
fmt:
    $(GOFMT)
lint:
    $(GOGET) github.com/golang/lint/golint
    golint
test:
    $(GOTEST) -v ./...
build:
    $(GOBUILD) -o $(BINARY_NAME) -v
run:
    $(GOBUILD) -o $(BINARY_NAME) -v ./...
    ./$(BINARY_NAME)
sonar:
    $(GOGET) -u $(SONAR_DEPENDENCIES)
    gometalinter --install
    -gometalinter --checkstyle > report.xml
    -gocov test ./... | gocov-xml > coverage.xml
    -$(GOTEST) -v ./... | go-junit-report > test.xml
    sonar-scanner -D sonar.host.url=http://192.168.0.11:9000 -D sonar.projectKey=helloworld -D sonar-sc
```

NOTE

- Each indent is a `tab` and **not** a series of `space` characters.
- Cutting-and-pasting the last line starting with `sonar-scanner` may paste this single line as multiple lines. It needs to be one single line in order to work.

Save the file and exit your editor.

Okay, let's try it out

```
make lint
make test
make build
make run
make sonar
```

The output will resemble

```
[vagrant@dev helloworld]$ make lint
go get github.com/golang/lint/golint
golint
[vagrant@dev helloworld]$ make test
go test -v ./...
=== RUN   TestHelloWorld
--- PASS: TestHelloWorld (0.00s)
PASS
ok      github.com/nemonik/helloworld   (cached)
[vagrant@dev helloworld]$ make build
go build -o helloworld -v
[vagrant@dev helloworld]$ make run
go build -o helloworld -v ./...
./helloworld
hello world
[vagrant@dev helloworld]$ make sonar
go get -u github.com/alecthomas/gometalinter github.com/axw/gocov/... github.com/AlekSi/gocov-xml github.com/jstemmer/go-
gometalinter --install
Installing:
  deadcode
  dupl
  errcheck
  gas
  goconst
  gocyclo
  goimports
  golint
  gosimple
  gotype
  gotypex
  ineffassign
  interfacer
  lll
  maligned
  megacheck
  misspell
  nakedret
  safesql
  staticcheck
  structcheck
  unconvert
  unparam
  unused
  varcheck
  vet
gometalinter --checkstyle > report.xml
gocov test ./... | gocov-xml > coverage.xml
ok      github.com/nemonik/helloworld   0.003s  coverage: 50.0% of statements
go test -v ./... | go-junit-report > test.xml
sonar-scanner -D sonar.host.url=http://192.168.0.11:9000 -D sonar.projectKey=helloworld -D sonar-scanner -D sonar.project
INFO: Scanner configuration file: /usr/local/sonar-scanner-3.1.0.1141-linux/conf/sonar-scanner.properties
INFO: Project root configuration file: NONE
INFO: SonarQube Scanner 3.1.0.1141
INFO: Java 1.8.0_121 Oracle Corporation (64-bit)
INFO: Linux 3.10.0-693.21.1.el7.x86_64 amd64
INFO: User cache: /home/vagrant/.sonar/cache
INFO: SonarQube server 7.0.0
INFO: Default locale: "en_US", source code encoding: "UTF-8" (analysis is platform dependent)
```

```
INFO: Publish mode
INFO: Load global settings
INFO: Load global settings (done) | time=92ms
INFO: Server id: AWLZ3KmeIXzDnCPHcQJq
INFO: User cache: /home/vagrant/.sonar/cache
INFO: Load plugins index
INFO: Load plugins index (done) | time=56ms
INFO: Load/download plugins
INFO: Load/download plugins (done) | time=6ms
INFO: Process project properties
INFO: Load project repositories
INFO: Load project repositories (done) | time=211ms
INFO: Load quality profiles
INFO: Load quality profiles (done) | time=129ms
INFO: Load active rules
INFO: Load active rules (done) | time=266ms
INFO: Load metrics repository
INFO: Load metrics repository (done) | time=51ms
WARN: SCM provider autodetection failed. No SCM provider claims to support this project. Please use sonar.scm.provider to
INFO: Project key: helloworld
INFO: -------------  Scan helloworld
INFO: Load server rules
INFO: Load server rules (done) | time=28ms
INFO: Base dir: /home/vagrant/go/src/github.com/nemonik/helloworld
INFO: Working dir: /home/vagrant/go/src/github.com/nemonik/helloworld/.scannerwork
INFO: Source paths: .
INFO: Source encoding: UTF-8, default locale: en_US
INFO: Index files
INFO: Excluded sources:
INFO:   **/*test.go
INFO: 6 files indexed
INFO: 1 file ignored because of inclusion/exclusion patterns
INFO: Quality profile for go: Golint Rules
INFO: Sensor GoMetaLinter issues loader sensor [golang]
INFO: Parsing the file report.xml
INFO: Parsing 'GoMetaLinter' Analysis Results
INFO: Sensor GoMetaLinter issues loader sensor [golang] (done) | time=22ms
INFO: Sensor Go Coverage [golang]
INFO: /home/vagrant/go/src/github.com/nemonik/helloworld
INFO: Analyse for /home/vagrant/go/src/github.com/nemonik/helloworld/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/refs/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/refs/heads/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/refs/tags/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/branches/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/hooks/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/info/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/objects/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/objects/pack/coverage.xml
INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld/.git/objects/info/coverage.xml
INFO: Sensor Go Coverage [golang] (done) | time=42ms
INFO: Sensor Go test JUnit loader sensor [golang]
INFO: base dir /home/vagrant/go/src/github.com/nemonik/helloworld
WARN: The key is null
INFO: Sensor Go test JUnit loader sensor [golang] (done) | time=5ms
INFO: Sensor Go Highlighter Sensor [golang]
INFO: Sensor Go Highlighter Sensor [golang] (done) | time=19ms
INFO: Sensor Go Metrics Sensor [golang]
INFO: Sensor Go Metrics Sensor [golang] (done) | time=2ms
INFO: Sensor Zero Coverage Sensor
```

```
INFO: Sensor Zero Coverage Sensor (done) | time=9ms
INFO: Sensor CPD Block Indexer
INFO: Sensor CPD Block Indexer (done) | time=1ms
INFO: No SCM system was detected. You can use the 'sonar.scm.provider' property to explicitly specify it.
INFO: Calculating CPD for 0 files
INFO: CPD calculation finished
INFO: Analysis report generated in 56ms, dir size=3 KB
INFO: Analysis reports compressed in 10ms, zip size=2 KB
INFO: Analysis report uploaded in 60ms
INFO: ANALYSIS SUCCESSFUL, you can browse http://192.168.0.11:9000/dashboard/index/helloworld
INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted analysis rep
INFO: More about the report processing at http://192.168.0.11:9000/api/ce/task?id=AWLaD8OGE0IcvTpxuSnt
INFO: Task total time: 1.604 s
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
INFO: Total time: 2.708s
INFO: Final Memory: 10M/116M
INFO: ------------------------------------------------------------------------```
```

8.9.12. Author Drone-based Continuous Integration

CI integrates all of the steps we have worked to ensure a high quality build into a pipeline, so let's do that.

We're going to author a continuous integration pipeline for our application and execute it on Drone. Drone expects a `.drone.yml` to exist at the root of the project and will execute the pipeline it contains when the project is committed to GitLab.

A pipeline is broken up into multiple named steps, where each step executes in an ephemeral Docker container with shared disk access to the project's workspace. The benefit of this approach is that it relieves you from having to create and maintain slaves to execute your pipelines.

Drone automatically clones your project's repo (Short for "repository.") into a volume (referred to as the workspace) shared by each Docker container (including plugins service containers).

In your editor create `.drone.yml` file (**Make sure you pre-pend that dot.**) at the root of the `helloworld` project:

```
pipeline:
  build:
    image: 192.168.0.11:5000/nemonik/golang:1.10.1
    commands:
      - make lint
      - make test
      - make build
  run:
    image: 192.168.0.11:5000/nemonik/golang:1.10.1
    commands:
      - make run
```

The pipeline is authored in YAML like almost all the CI orchestrators out there except for Jenkin's Pipelines, whose you author in Groovy-based DSL.

- `pipeline:` - defines a list of steps to build, test and deploy your code.

- `build:` and `run:` - are the names of the step. These are yours to name. Name steps something meaningful as to what the step is orchestrating. Each step is executed serially, in the order defined.
- `image: 192.168.0.11:5000/nemonik/golang:1.10.1` - The build step is using the nemonik/golang container tagged `1.10.1` retrieved from private Docker registry located at `192.168.0.11:5000`. Drone uses Docker images for the build environment, plugins and service containers. Drone spins them up for the execution of the pipeline and when no longer needed they go poof.
- `commands` - Is a collection of terminal commands to be executed. These are all the same commands we executed previously in the command line. If anyone of these commands were to fail returning a non-zero exit code, the pipeline will immediately end resulting in a failed build.

8.9.13. Configure Drone to execute your pipeline

1. Open http://192.168.0.11:8000/ in your browser and authenticate through GitLab on into Drone, if you need to.
2. In the upper right corner find the hamburger icon (displayed as ☰) and click it and then select `Repositories`.
3. Then click hamburger icon again and select `Synchronize`, if you don't see your project's repo.
4. Then click the slider to for the `root/helloworld` repo to enable Drone orchestration for the project.
5. Clicking the box icon with the arrow pointing toward upper-right to open root/helloworld's builds.

You won't have any builds to start, but when you do the builds will increment starting from 1. Red is failed the build. Yellow-orange is a presently executing build. Green is a build that passed. When a build does start, click on its row to open and monitor it. The UI will update as the build proceeds informing you as to its progress.

8.9.14. Trigger the build

To trigger the build, in your ssh connection to `dev` simply commit your code:

```
git add .
git commit –m "Added Drone pipeline"
git push –u origin master
```

Immediately after you enter your GitLab username/password open http://192.168.0.11:8000/root/hello in your browser, if you re-use an existing tab to this page refresh the page.

The execution of this pipeline will follow as so:

1. A new build will appear. Click on it.
2. Drone will clone your project's repository in a `clone` step.
3. And then spin up `nemonik/golang:1.9.4` container, whose container image was patched to work behind MITRE's http proxy and SSL introspection. (Later, you may want examine the contents of ansible/roles/golang-container-image/files/Dockerfile for more details as to how this was accomplished.)
4. And then execute the following shell command in the same way you executed them yourself: a. make lint b. make test c. make build d. make run

The output of the `build` (An arbitrary name. You could use "skippy".) step will resemble:

```
+ make lint
go get github.com/golang/lint/golint
golint
+ make test
go test -v ./...
=== RUN   TestHelloWorld
--- PASS: TestHelloWorld (0.00s)
PASS
ok      _/drone/src/192.168.0.11/root/helloworld        0.003s
+ make build
go build -o helloworld -v
_/drone/src/192.168.0.11/root/helloworld
```

The output of the `run` step will resemble:

```
+ make run
go build -o helloworld -v ./...
./helloworld
hello world
```

Our build was successful. Drone uses a container's exit code to determine success or failure. A container's non-zero exit code will cause the pipeline to exit immediately.

That's it. This is essentially CI.

8.9.15. The completed source for *helloworld*

The `helloworld` project can be viewed completed at

https://gitlab.mitre.org/mjwalsh/helloworld

8.10. Golang *helloworld-web* project

Like `helloworld`, the `helloworld-web` project is a very simple application that we will use to explore Continuous Deliver. Remember, Continuous Delivery builds upon Continuous Integration.

8.10.1. Create the project's backlog

Open Taiga in your web browser

http://192.168.0.11:8080

Complete the follow to track your progress in completing the *helloworld-web* project

1. Click `Create Project`.
2. Select `Kanban`. A Kanban board shows how work moves from left to right, each column represents a stage within the value stream.
3. Give your project a name. For example, `Helloworld-web` and a description, such as, `My Kanban board for this awesome helloworld-web app` and then click `CREATE PROJECT`.

4. You can skip this step and opt to chose to click `><` to fold `READY`, `USER STORY STATUS` and `ARCHIVED` only after completing step 6. Otherwise, you can edit your Kanban board to just `NEW`, `IN PROGRESS`, and `DONE` by
   a. On the bottom left, click the `ADMIN` gear.
   b. Click `ATTRIBUTES`.
   c. Scroll down to `USER STORY STATUS`.
   d. Hover over `Ready`, click the trash icon to delete and click `ACCEPT`.
   c. Do the same for `Ready for test` and `Archived`.
   d. Click the `KANBAN` icon on the far left. It Looks like columns. And then reload the browser to get the changes to take
5. In the `NEW` column select `Add New bulk` icon that looks like a list and when the page updates cut-and-paste the lines below into the text box and click `SAVE`.

```
Create the project in GitLab
Create the project
Retrieve dependencies
Author the application
Build and Run the application
Run sonar-scanner on the application
Write the Makefile
Dockerize the application
Run the container
Push the container image to the private registry
Author the build step for the pipeline
Author the publish step for the pipeline
Add a deploy step to the pipeline
Add SonarQube to the pipeline
Author an automated functional test
Add a selenium step to our pipeline
Add a DAST step for the pipeline
```

Track your progress in Taiga as you work through each section.

8.10.2. Create the project in GitLab

1. In GitLab (http://192.168.0.11:10080/) click on `Projects` in the upper left.
   a. Select `Your projects` from the dropdown.
   b. Click the green `New Project` on the far right under `Projects`, or
   c. Click http://192.168.0.11:10080/projects/new
2. Leave the `Project path` defaulted to `http://192.168.0.11:10080/root/`.
3. Enter `helloworld-web` for the `Project name`. **Be careful with the spelling.**
4. Provide an optional `Project description`. Something descriptive, such as, "*GoLang helloworld application for the hands-on DevOps class.*".
5. Make the application `public`.
6. Click the green `Create Project` button on the lower left.

The UI will refresh to show you landing page for the project that should be accessible from

http://192.168.0.11:10080/root/helloworld-web

8.10.3. Setup your project on the *dev* Vagrant

Create a `helloworld-web` GitLab hosted repo as you did prior for the `helloworld` project.

On your host in secure shell to `dev` vagrant by running at the root of the class project

```
vagrant ssh dev
```

If you had just ented `vagrant ssh` you will be connected to the `dev` vagrant by default.

Then in the command line enter

```
cd ~/go/src/github.com/nemonik
git clone http://192.168.0.11:10080/root/helloworld-web.git
cd helloworld-web
```

NOTE

- The `git clone` will fail if you did not name your project correctly while in GitLab.

So that you do not commit certian files to GitLab when you push, create a `.gitignore` file with your editor with this content

```
# OS-specific
.DS_Store

*.xml

helloworld-web

.scannerwork
```

8.10.4. Author the application

Create `main.go` in emacs, nano, vi, or vim with this content:

```
package main

import (
        "fmt"
        "net/http"
)

func main() {
        http.HandleFunc("/", handler)
        fmt.Print("listening on :3000\n")
        http.ListenAndServe(":3000", logRequest(http.DefaultServeMux))
}

func handler(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintf(w, "Hello world!")
}

func logRequest(handler http.Handler) http.Handler {
        return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
                fmt.Printf("%s  %s  %s\n", r.RemoteAddr, r.Method, r.URL)
                handler.ServeHTTP(w, r)
        })
}
```

8.10.5. Build and Run the application

Run the application

```
go build -o helloworld-web .
```

No output means things are peachy. Otherwise fix your mistakes.

Now run

```
./helloworld-web
```

Command line output will be

```
[vagrant@dev helloworld-web]$ ./helloworld-web
listening on :3000
```

Open in a web browser:

http://192.168.0.10:3000

The browser will load

```
Hello world!
```

The command line will output

```
192.168.0.1:63201  GET  /
192.168.0.1:63201  GET  /favicon.ico
```

8.10.6. Run sonar-scanner on the application

The following command line steps are optional as you have installed these dependencies for the `helloworld` application

```
go get github.com/alecthomas/gometalinter
go get github.com/axw/gocov/...
go get github.com/AlekSi/gocov-xml
go get github.com/jstemmer/go-junit-report
gometalinter --install
```

But these must be entered to generate the reports and submit to SonarQube

```
gometalinter --checkstyle > report.xml
gocov test ./... | gocov-xml > coverage.xml
go test -v ./... | go-junit-report > test.xml
sonar-scanner -D sonar.projectKey=helloworld-web -D sonar-scanner -D sonar.projectKey=helloworld-web -D sonar.projectName
```

Open in your host's web browser

http://192.168.0.11:9000/dashboard?id=helloworld-web

to view your SonarQube report.

SonarQube has picked up we have no code coverage (i.e., unit test.)

8.10.7. Let's write some unit test to up that coverage

So, not writing any unit tests prior to running Sonar we had code coverage issues.

Let's fix that.

Create `main_test.go` in emacs, nano, vi, or vim with this content:

```
package main

import (
        "net/http"
        "net/http/httptest"
        "os"
        "testing"
)

func TestHandler(t *testing.T) {
        // Create an http request
        req, _ := http.NewRequest("GET", "/", nil)

        // Create  http.ResponseWriter for test inspection
        w := httptest.NewRecorder()

        // Call the handler
        handler(w, req)

        // Inspect the http.ResponseWriter
        if w.Code != http.StatusOK {
                t.Errorf("Server did not return %v", http.StatusOK)
        }

        if w.Body.String() != "Hello world!" {
                t.Errorf("Body contain \"%v\" instead of expected \"Hello world!\"", w.Body.String())
        }
}

func TestMain(m *testing.M) {
        os.Exit(m.Run())
}
```

Execute the unit test by entering

```
go test
```

The command line returns

```
[vagrant@dev helloworld-web]$ go test
PASS
ok      github.com/nemonik/helloworld-web       0.005s
```

Re-generate the reports and submit to SonarQube

```
gometalinter --checkstyle > report.xml
gocov test ./... | gocov-xml > coverage.xml
go test -v ./... | go-junit-report > test.xml
sonar-scanner -D sonar.projectKey=helloworld-web -D sonar-scanner -D sonar.projectKey=helloworld-web -D sonar.projectName
```

Open in your host's web browser

http://192.168.0.11:9000/dashboard?id=helloworld-web

to view your SonarQube report.

SonarQube has picked up we have better code coverage. Better, but not perfect.

8.10.8. Write the Makefile

In your editor create a `Makefile` to ensure the build and the steps leading to are repeatable.

```
GOCMD=go
GOFMT=$(GOCMD) fmt
GOBUILD=$(GOCMD) build
GOCLEAN=$(GOCMD) clean
GOTEST=$(GOCMD) test
GOGET=$(GOCMD) get
BINARY_NAME=helloworld-web
HTTP_PROXY=${http_proxy}
NO_PROXY=${no_proxy}

SONAR_DEPENDENCIES := \
        github.com/alecthomas/gometalinter \
        github.com/axw/gocov/... \
        github.com/AlekSi/gocov-xml \
        github.com/jstemmer/go-junit-report

all: fmt lint test build
clean:
        $(GOCLEAN)
        rm -f $(BINARY_NAME)
fmt:
        $(GOFMT)
lint:
        $(GOGET) github.com/golang/lint/golint
        golint
test:
        $(GOTEST) -v ./...
build:
        $(GOBUILD) -o $(BINARY_NAME) -v
run:
        $(GOBUILD) -o $(BINARY_NAME) -v ./...
        ./$(BINARY_NAME)
sonar:
        $(GOGET) $(SONAR_DEPENDENCIES)
        gometalinter --install
        -gometalinter --checkstyle > report.xml
        -gocov test ./... | gocov-xml > coverage.xml
        -$(GOTEST) -v ./... | go-junit-report > test.xml
        sonar-scanner -X -D http.nonProxyHosts=$(NO_PROXY) -D http.proxyHost=$(HTTP_PROXY) -D sonar.hos
```

Looks more or less like `helloworld` 's Makefile with the addition of `deps` .

Again, remember

- The indents are `tab` characters and not `spaces` characters.
- And cut-and-paste may split the single line beginning with `sonar-scanner` into multiple lines. You want this to be single line of text.

Test out your `Makefile`

```
make lint
make test
make build
make sonar
make run
```

And so, you have build and test automation whose output resembles

```
[vagrant@dev helloworld-web]$ make lint
go get github.com/golang/lint/golint
golint
[vagrant@dev helloworld-web]$ make test
go test -v ./...
=== RUN   TestHandler
--- PASS: TestHandler (0.00s)
PASS
ok      github.com/nemonik/helloworld-web       (cached)
[vagrant@dev helloworld-web]$ make build
go build -o helloworld-web -v
[vagrant@dev helloworld-web]$ make sonar
go get github.com/alecthomas/gometalinter github.com/axw/gocov/... github.com/AlekSi/gocov-xml github.com/jstemmer/go-jun
gometalinter --install
Installing:
  deadcode
  dupl
  errcheck
  gas
  goconst
  gocyclo
  goimports
  golint
  gosimple
  gotype
  gotypex
  ineffassign
  interfacer
  lll
  maligned
  megacheck
  misspell
  nakedret
  safesql
  staticcheck
  structcheck
  unconvert
  unparam
  unused
  varcheck
  vet
gometalinter --checkstyle > report.xml
make: [sonar] Error 1 (ignored)
gocov test ./... | gocov-xml > coverage.xml
ok      github.com/nemonik/helloworld-web       0.005s  coverage: 14.3% of statements
go test -v ./... | go-junit-report > test.xml
sonar-scanner -X -D http.nonProxyHosts=127.0.0.1,localhost,.mitre.org,.local,192.168.0.1,192.168.0.2,192.168.0.3,192.168.
20:05:58.469 INFO: Scanner configuration file: /usr/local/sonar-scanner-3.1.0.1141-linux/conf/sonar-scanner.properties
20:05:58.476 INFO: Project root configuration file: NONE
20:05:58.517 INFO: SonarQube Scanner 3.1.0.1141
20:05:58.517 INFO: Java 1.8.0_121 Oracle Corporation (64-bit)
20:05:58.517 INFO: Linux 3.10.0-693.21.1.el7.x86_64 amd64
20:05:58.710 DEBUG: keyStore is :
20:05:58.710 DEBUG: keyStore type is : jks
20:05:58.710 DEBUG: keyStore provider is :
20:05:58.710 DEBUG: init keystore
20:05:58.710 DEBUG: init keymanager of type SunX509
20:05:58.807 DEBUG: Create: /home/vagrant/.sonar/cache
```

```
20:05:58.808 INFO: User cache: /home/vagrant/.sonar/cache
20:05:58.808 DEBUG: Create: /home/vagrant/.sonar/cache/_tmp
20:05:58.810 DEBUG: Extract sonar-scanner-api-batch in temp...
20:05:58.818 DEBUG: Get bootstrap index...
20:05:58.818 DEBUG: Download: http://192.168.0.11:9000/batch/index
20:05:58.885 DEBUG: Get bootstrap completed
20:05:58.894 DEBUG: Create isolated classloader...
20:05:58.903 DEBUG: Start temp cleaning...
20:05:58.911 DEBUG: Temp cleaning done
20:05:58.912 DEBUG: Execution getVersion
20:05:58.924 INFO: SonarQube server 7.0.0
20:05:58.926 INFO: Default locale: "en_US", source code encoding: "UTF-8" (analysis is platform dependent)
20:05:58.927 DEBUG: Work directory: /home/vagrant/go/src/github.com/nemonik/helloworld-web/.scannerwork
20:05:58.928 DEBUG: Execution execute
20:05:59.164 INFO: Publish mode
20:05:59.301 INFO: Load global settings
20:05:59.351 DEBUG: GET 200 http://192.168.0.11:9000/api/settings/values.protobuf | time=40ms
20:05:59.374 INFO: Load global settings (done) | time=74ms
20:05:59.380 INFO: Server id: AWLZ3KmeIXzDnCPHcQJq
20:05:59.392 INFO: User cache: /home/vagrant/.sonar/cache
20:05:59.584 INFO: Load plugins index
20:05:59.591 DEBUG: GET 200 http://192.168.0.11:9000/api/plugins/installed | time=7ms
20:05:59.626 INFO: Load plugins index (done) | time=41ms
20:05:59.626 INFO: Load/download plugins
20:05:59.633 INFO: Load/download plugins (done) | time=7ms
20:05:59.652 DEBUG: Plugins:
20:05:59.652 DEBUG:   * Golang 1.2.11-rc13 (golang)
20:05:59.919 INFO: Process project properties
20:05:59.926 DEBUG: Process project properties (done) | time=7ms
20:05:59.950 INFO: Load project repositories
20:05:59.964 DEBUG: GET 200 http://192.168.0.11:9000/batch/project.protobuf?key=helloworld-web | time=11ms
20:06:00.027 INFO: Load project repositories (done) | time=77ms
20:06:00.103 DEBUG: Available languages:
20:06:00.103 DEBUG:   * GO => "go"
20:06:00.114 INFO: Load quality profiles
20:06:00.131 DEBUG: GET 200 http://192.168.0.11:9000/api/qualityprofiles/search.protobuf?projectKey=helloworld-web | time
20:06:00.137 INFO: Load quality profiles (done) | time=23ms
20:06:00.152 INFO: Load active rules
20:06:00.194 DEBUG: GET 200 http://192.168.0.11:9000/api/rules/search.protobuf?f=repo,name,severity,lang,internalKey,temp
20:06:00.241 INFO: Load active rules (done) | time=88ms
20:06:00.245 INFO: Load metrics repository
20:06:00.255 DEBUG: GET 200 http://192.168.0.11:9000/api/metrics/search?f=name,description,direction,qualitative,custom&p
20:06:00.271 INFO: Load metrics repository (done) | time=26ms
20:06:00.282 WARN: SCM provider autodetection failed. No SCM provider claims to support this project. Please use sonar.sc
20:06:00.287 INFO: Project key: helloworld-web
20:06:00.287 DEBUG: Start recursive analysis of project modules
20:06:00.288 INFO: -------------  Scan helloworld-web
20:06:00.354 INFO: Load server rules
20:06:00.361 DEBUG: GET 200 http://192.168.0.11:9000/api/rules/list.protobuf | time=6ms
20:06:00.365 INFO: Load server rules (done) | time=11ms
20:06:00.394 INFO: Base dir: /home/vagrant/go/src/github.com/nemonik/helloworld-web
20:06:00.394 INFO: Working dir: /home/vagrant/go/src/github.com/nemonik/helloworld-web/.scannerwork
20:06:00.396 INFO: Source paths: .
20:06:00.396 INFO: Source encoding: UTF-8, default locale: en_US
20:06:00.418 DEBUG: Declared extensions of language GO were converted to sonar.lang.patterns.go : **/*.go
20:06:00.422 DEBUG: Initializers :
20:06:00.424 INFO: Index files
20:06:00.431 INFO: Excluded sources:
20:06:00.431 INFO:   **/*test.go
```

```
20:06:00.441 DEBUG: 'main.go' indexed with language 'go'
20:06:00.443 DEBUG: 'helloworld-web' indexed with language 'null'
20:06:00.445 DEBUG: 'report.xml' indexed with language 'null'
20:06:00.446 DEBUG: 'coverage.xml' indexed with language 'null'
20:06:00.446 DEBUG: 'test.xml' indexed with language 'null'
20:06:00.446 DEBUG: 'Makefile' indexed with language 'null'
20:06:00.448 INFO: 6 files indexed
20:06:00.449 INFO: 1 file ignored because of inclusion/exclusion patterns
20:06:00.449 INFO: Quality profile for go: Golint Rules
20:06:00.459 DEBUG: 'Generic Coverage Report' skipped because one of the required properties is missing
20:06:00.459 DEBUG: 'Generic Test Executions Report' skipped because one of the required properties is missing
20:06:00.468 DEBUG: 'Generic Coverage Report' skipped because one of the required properties is missing
20:06:00.469 DEBUG: 'Generic Test Executions Report' skipped because one of the required properties is missing
20:06:00.472 DEBUG: Sensors : GoMetaLinter issues loader sensor -> Go Coverage -> Go test JUnit loader sensor -> Go Highl
20:06:00.473 INFO: Sensor GoMetaLinter issues loader sensor [golang]
20:06:00.511 INFO: Parsing the file report.xml
20:06:00.511 INFO: Parsing 'GoMetaLinter' Analysis Results
20:06:00.512 DEBUG: Parsing file /home/vagrant/go/src/github.com/nemonik/helloworld-web/report.xml
20:06:00.533 DEBUG: violation found for the file main.go
20:06:00.534 INFO: Load /key.properties
20:06:00.535 INFO: loaded 52
20:06:00.551 DEBUG: 'main.go' generated metadata  with charset 'UTF-8'
20:06:00.565 INFO: Sensor GoMetaLinter issues loader sensor [golang] (done) | time=92ms
20:06:00.565 INFO: Sensor Go Coverage [golang]
20:06:00.565 INFO: /home/vagrant/go/src/github.com/nemonik/helloworld-web
20:06:00.569 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web
20:06:00.569 INFO: Analyse for /home/vagrant/go/src/github.com/nemonik/helloworld-web/coverage.xml
20:06:00.580 DEBUG: filepath from coverage file /home/vagrant/go/src/github.com/nemonik/helloworld-web/main.go
20:06:00.581 DEBUG: 7line coverage for file /home/vagrant/go/src/github.com/nemonik/helloworld-web/main.go
20:06:00.595 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/refs
20:06:00.595 INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/refs/coverage.
20:06:00.595 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/refs/heads
20:06:00.595 INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/refs/heads/cov
20:06:00.597 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/refs/tags
20:06:00.597 INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/refs/tags/cove
20:06:00.597 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/branches
20:06:00.597 INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/branches/cover
20:06:00.597 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/description
20:06:00.598 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/hooks
20:06:00.598 INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/hooks/coverage
20:06:00.598 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/hooks/applypatch-msg.sample
20:06:00.598 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/hooks/commit-msg.sample
20:06:00.599 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/hooks/post-update.sample
20:06:00.599 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/hooks/pre-applypatch.sample
20:06:00.599 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/hooks/pre-commit.sample
20:06:00.599 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/hooks/pre-push.sample
20:06:00.599 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/hooks/pre-rebase.sample
20:06:00.600 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/hooks/prepare-commit-msg.sa
20:06:00.600 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/hooks/update.sample
20:06:00.600 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/info
20:06:00.600 INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/info/coverage.
20:06:00.600 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/info/exclude
20:06:00.601 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/HEAD
20:06:00.601 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/config
20:06:00.601 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/objects
20:06:00.601 INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/objects/covera
20:06:00.601 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/objects/pack
20:06:00.602 INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/objects/pack/c
20:06:00.602 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/objects/info
```

```
20:06:00.602 INFO: no coverage file in package /home/vagrant/go/src/github.com/nemonik/helloworld-web/.git/objects/info/c
20:06:00.602 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/main.go
20:06:00.602 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/helloworld-web
20:06:00.603 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/report.xml
20:06:00.603 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/coverage.xml
20:06:00.604 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/test.xml
20:06:00.604 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/main_test.go
20:06:00.604 DEBUG: Path in stream/home/vagrant/go/src/github.com/nemonik/helloworld-web/Makefile
20:06:00.607 INFO: Sensor Go Coverage [golang] (done) | time=42ms
20:06:00.607 INFO: Sensor Go test JUnit loader sensor [golang]
20:06:00.608 INFO: base dir /home/vagrant/go/src/github.com/nemonik/helloworld-web
20:06:00.610 DEBUG: {}
20:06:00.610 DEBUG: {}
20:06:00.613 WARN: The key is null
20:06:00.614 INFO: Sensor Go test JUnit loader sensor [golang] (done) | time=6ms
20:06:00.614 INFO: Sensor Go Highlighter Sensor [golang]
20:06:00.615 DEBUG: Coloring the file: /home/vagrant/go/src/github.com/nemonik/helloworld-web/main.go
20:06:00.628 DEBUG: Line number 1 index start: 0 index end: 7
20:06:00.629 DEBUG: Line number 3 index start: 0 index end: 6
20:06:00.629 DEBUG:     "fmt" 1
20:06:00.629 DEBUG:     "fmt" 5
20:06:00.629 DEBUG: index 1 indexEnd 5
20:06:00.629 DEBUG:     "fmt" -1
20:06:00.629 DEBUG:     "fmt" 1
20:06:00.630 DEBUG:     "net/http" 1
20:06:00.630 DEBUG:     "net/http" 10
20:06:00.630 DEBUG: index 1 indexEnd 10
20:06:00.630 DEBUG:     "net/http" -1
20:06:00.630 DEBUG:     "net/http" 1
20:06:00.630 DEBUG: Line number 8 index start: 0 index end: 4
20:06:00.630 DEBUG:     http.HandleFunc("/", handler) 17
20:06:00.630 DEBUG:     http.HandleFunc("/", handler) 19
20:06:00.630 DEBUG: index 17 indexEnd 19
20:06:00.630 DEBUG:     http.HandleFunc("/", handler) -1
20:06:00.630 DEBUG:     http.HandleFunc("/", handler) 17
20:06:00.630 DEBUG:     fmt.Print("listening on :3000\n") 11
20:06:00.630 DEBUG:     fmt.Print("listening on :3000\n") 32
20:06:00.630 DEBUG: index 11 indexEnd 32
20:06:00.631 DEBUG:     fmt.Print("listening on :3000\n") -1
20:06:00.631 DEBUG:     fmt.Print("listening on :3000\n") 11
20:06:00.631 DEBUG: Line number 10 index start: 7 index end: 10
20:06:00.631 DEBUG:     http.ListenAndServe(":3000", logRequest(http.DefaultServeMux)) 21
20:06:00.631 DEBUG:     http.ListenAndServe(":3000", logRequest(http.DefaultServeMux)) 27
20:06:00.631 DEBUG: index 21 indexEnd 27
20:06:00.631 DEBUG:     http.ListenAndServe(":3000", logRequest(http.DefaultServeMux)) -1
20:06:00.631 DEBUG:     http.ListenAndServe(":3000", logRequest(http.DefaultServeMux)) 21
20:06:00.631 DEBUG: Line number 14 index start: 0 index end: 4
20:06:00.631 DEBUG:     fmt.Fprintf(w, "Hello world!") 16
20:06:00.632 DEBUG:     fmt.Fprintf(w, "Hello world!") 29
20:06:00.632 DEBUG: index 16 indexEnd 29
20:06:00.632 DEBUG:     fmt.Fprintf(w, "Hello world!") -1
20:06:00.632 DEBUG:     fmt.Fprintf(w, "Hello world!") 16
20:06:00.632 DEBUG: Line number 15 index start: 8 index end: 11
20:06:00.632 DEBUG: Line number 18 index start: 0 index end: 4
20:06:00.632 DEBUG: Line number 19 index start: 25 index end: 29
20:06:00.632 DEBUG: Line number 19 index start: 1 index end: 7
20:06:00.632 DEBUG:             fmt.Printf("%s  %s  %s\n", r.RemoteAddr, r.Method, r.URL) 13
20:06:00.632 DEBUG:             fmt.Printf("%s  %s  %s\n", r.RemoteAddr, r.Method, r.URL) 26
20:06:00.632 DEBUG: index 13 indexEnd 26
```

```
20:06:00.632 DEBUG:             fmt.Printf("%s  %s  %s\n", r.RemoteAddr, r.Method, r.URL) -1
20:06:00.633 DEBUG:             fmt.Printf("%s  %s  %s\n", r.RemoteAddr, r.Method, r.URL) 13
20:06:00.633 DEBUG: Line number 20 index start: 8 index end: 11
20:06:00.639 INFO: Sensor Go Highlighter Sensor [golang] (done) | time=25ms
20:06:00.639 INFO: Sensor Go Metrics Sensor [golang]
20:06:00.641 INFO: Sensor Go Metrics Sensor [golang] (done) | time=2ms
20:06:00.641 INFO: Sensor Zero Coverage Sensor
20:06:00.651 INFO: Sensor Zero Coverage Sensor (done) | time=10ms
20:06:00.651 INFO: Sensor CPD Block Indexer
20:06:00.652 DEBUG: org.sonar.scanner.cpd.deprecated.DefaultCpdBlockIndexer is used for go
20:06:00.652 DEBUG: No CpdMapping for language go
20:06:00.652 INFO: Sensor CPD Block Indexer (done) | time=1ms
20:06:00.652 INFO: No SCM system was detected. You can use the 'sonar.scm.provider' property to explicitly specify it.
20:06:00.653 INFO: Calculating CPD for 0 files
20:06:00.655 INFO: CPD calculation finished
20:06:00.717 INFO: Analysis report generated in 61ms, dir size=20 KB
20:06:00.730 INFO: Analysis reports compressed in 12ms, zip size=6 KB
20:06:00.730 INFO: Analysis report generated in /home/vagrant/go/src/github.com/nemonik/helloworld-web/.scannerwork/scann
20:06:00.730 DEBUG: Upload report
20:06:00.753 DEBUG: POST 200 http://192.168.0.11:9000/api/ce/submit?projectKey=helloworld-web&projectName=helloworld-web
20:06:00.755 INFO: Analysis report uploaded in 24ms
20:06:00.756 INFO: ANALYSIS SUCCESSFUL, you can browse http://192.168.0.11:9000/dashboard/index/helloworld-web
20:06:00.756 INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted
20:06:00.756 INFO: More about the report processing at http://192.168.0.11:9000/api/ce/task?id=AWLaXScEE0IcvTpxuSn-
20:06:00.757 DEBUG: Report metadata written to /home/vagrant/go/src/github.com/nemonik/helloworld-web/.scannerwork/report
20:06:00.759 DEBUG: Post-jobs :
20:06:00.764 INFO: Task total time: 1.110 s
20:06:00.833 INFO: ------------------------------------------------------------------------
20:06:00.833 INFO: EXECUTION SUCCESS
20:06:00.833 INFO: ------------------------------------------------------------------------
20:06:00.833 INFO: Total time: 2.402s
20:06:00.884 INFO: Final Memory: 10M/117M
20:06:00.884 INFO: ------------------------------------------------------------------------
[vagrant@dev helloworld-web]$ make run
go build -o helloworld-web -v ./...
./helloworld-web
listening on :3000
^Cmake: *** [run] Interrupt
```

8.10.9. Dockerize the application

Build a Docker image named `nemonik/helloworld-web` for the application and all its dependencies to ensure a repeatable deployment on Docker by opening the file `Dockerfile` in an editor at the root of the project and copy the is content into it

```
FROM 192.168.0.11:5000/nemonik/golang:1.10.1
MAINTAINER Michael Joseph Walsh <nemonik@gmail.com>

RUN mkdir /app
ADD main.go /app/

WORKDIR /app

RUN go build -o helloworld-web .

CMD ["/app/helloworld-web"]

EXPOSE 3000
```

Then create the Docker image

`docker build -t nemonik/helloworld-web .` Don't miss that last period (`.`).

Command line output will resemble

```
[vagrant@dev helloworld-web]$ docker build -t nemonik/helloworld-web .
Sending build context to Docker daemon  6.704MB
Step 1/8 : FROM 192.168.0.11:5000/nemonik/golang:1.10.1
1.10.1: Pulling from nemonik/golang
c73ab1c6897b: Pull complete
1ab373b3deae: Pull complete
b542772b4177: Pull complete
57c8de432dbe: Pull complete
c81227e1ec90: Pull complete
df4d04d0f54c: Pull complete
8ed488a16a31: Pull complete
0828adce2551: Pull complete
9b8c7e5da5a1: Pull complete
0b398c7e2292: Pull complete
30e8e73c0c04: Pull complete
1de129fb171b: Pull complete
27eab3e938d4: Pull complete
17d8f862a4a2: Pull complete
38a4faa53d09: Pull complete
Digest: sha256:80864483daf76493863de6388a0bee977ddbc002d7b0cb19f253b3a1ddc8e14b
Status: Downloaded newer image for 192.168.0.11:5000/nemonik/golang:1.10.1
 ---> 7c239dc2f416
Step 2/8 : MAINTAINER Michael Joseph Walsh <nemonik@gmail.com>
 ---> Running in c90fdce9e639
Removing intermediate container c90fdce9e639
 ---> b3aee577a01b
Step 3/8 : RUN mkdir /app
 ---> Running in 24a817dd0a33
Removing intermediate container 24a817dd0a33
 ---> 8be878c3b7d0
Step 4/8 : ADD main.go /app/
 ---> ba34e7e243ce
Step 5/8 : WORKDIR /app
Removing intermediate container 763c31ec2c24
 ---> dccda18cc25f
Step 6/8 : RUN go build -o helloworld-web .
 ---> Running in 8e75c766891d
Removing intermediate container 8e75c766891d
 ---> 39414f81a3dc
Step 7/8 : CMD ["/app/helloworld-web"]
 ---> Running in 73a1b6ac25e5
Removing intermediate container 73a1b6ac25e5
 ---> 65892d7e6440
Step 8/8 : EXPOSE 3000
 ---> Running in 852b23e5dbd0
Removing intermediate container 852b23e5dbd0
 ---> 5f47437b3b2a
Successfully built 5f47437b3b2a
Successfully tagged nemonik/helloworld-web:latest
```

`docker build` places the image in the local Docker registry so that containers can be created locally. Check the local docker registry for

```
[vagrant@dev helloworld-web]$ docker images
```

Command line output will resemble

```
[vagrant@dev helloworld-web]$ docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
nemonik/helloworld-web              latest              5f47437b3b2a        53 seconds ago      835MB
192.168.0.11:5000/nemonik/golang    1.10.1              7c239dc2f416        2 hours ago         828MB
```

We can update the project's Makefile adding a `docker-build` target

```
docker-build:
        docker build --no-cache -t nemonik/helloworld-web .
```

Remember to us tab characters vice space characters.

And then run in the command lin

```
make docker-build
```

8.10.10. Run the container

Spin up a new `nemonik/helloworld-web` container by entering

```
docker run -p 3000:3000 nemonik/helloworld-web
```

Where

- `run` messages docker you are running a new container
- `-p 3000:3000` published the container's port as 3000 to the host.
- `nemonik/helloworld-web` states what container image to use.

The command line output will be

```
[vagrant@dev helloworld-web]$ docker run -p 3000:3000 nemonik/helloworld-web
listening on :3000 (
```

`ctrl-c` in the command line to stop the container.

8.10.11. Push the container image to the private registry

Push the `nemonik/hellowrold-web` container image into the private Docker registry running on the `Toolchain` vagrant, so that both vagrants can create containers from the image with the commands

```
docker tag nemonik/helloworld-web 192.168.0.11:5000/nemonik/helloworld-web
docker push 192.168.0.11:5000/nemonik/helloworld-web
```

Command line output will be

```
[vagrant@dev helloworld-web]$ docker push 192.168.0.11:5000/nemonik/helloworld-web
The push refers to repository [192.168.0.11:5000/nemonik/helloworld-web]
733e19430470: Pushed
72b90bd18970: Pushed
65e96847b487: Pushed
5f69028652b8: Mounted from nemonik/golang
c6d8b0b39323: Mounted from nemonik/golang
a77a1a834917: Mounted from nemonik/golang
92ce06e068ca: Mounted from nemonik/golang
ab8d7a56d156: Mounted from nemonik/golang
30af257ded3b: Mounted from nemonik/golang
120f2e939f3b: Mounted from nemonik/golang
47823918137b: Mounted from nemonik/golang
2bd1ea5f0540: Mounted from nemonik/golang
f2ab791795ee: Mounted from nemonik/golang
e33f0ab4ff77: Mounted from nemonik/golang
20c527f217db: Mounted from nemonik/golang
61c06e07759a: Mounted from nemonik/golang
bcbe43405751: Mounted from nemonik/golang
e1df5dc88d2c: Mounted from nemonik/golang
latest: digest: sha256:59d026c810f0bec333674df6b19e3b8a8fc12c922843579147559996edc6f61a size: 4101
```

We can update the project's Makefile adding a `docker-push` target

```
docker-push:
	docker tag nemonik/helloworld-web 192.168.0.11:5000/nemonik/helloworld-web
	docker push 192.168.0.11:5000/nemonik/helloworld-web
```

And then run in the command line

```
make docker-push
```

The Docker registry container image shipped by Docker does not provide a GUI, but we can verify by querying the catalog of the private registry through a web browser or Unix command line tool `curl` by entering into the command line

```
curl -X GET http://192.168.0.11:5000/v2/_catalog
```

Returns in the command line

```
[vagrant@dev helloworld-web]$ curl -X GET http://192.168.0.11:5000/v2/_catalog
{"repositories":["nemonik/golang","nemonik/golang-sonarqube-scanner","nemonik/helloworld-web","nemonik/python","nemonik/s
```

Query for the container you pushed

```
curl -X GET http://192.168.0.11:5000/v2/nemonik/helloworld-web/tags/list
```

Returns in the command line

```
[vagrant@dev helloworld-web]$ curl -X GET http://192.168.0.11:5000/v2/nemonik/helloworld-web/tags/list
{"name":"nemonik/helloworld-web","tags":["latest"]}
```

8.10.12. Configure Drone to execute your pipeline

1. Open http://192.168.0.11:8000/ in your browser and authenticate through GitLab on into Drone, if you need to.
2. In the upper right corner find the hamburger icon (displayed as ☰) and click it and then select `Repositories`.
3. Then click hamburger icon again and select `Synchronize`, if you don't see your project's repo.
4. Then click the slider to for the `root/helloworld-web` repo to enable Drone orchestration for the project.
5. Clicking the box icon with the arrow pointing toward upper-right to open root/helloworld-web's builds.

8.10.13. Author the build step for the pipeline

At the root of the project open `.drone.yml` as a new file to edit

```
pipeline:
  build:
    image: nemonik/golang:1.10.1
    commands:
      - make lint
      - make test
      - make build
```

Then commit to GitLab

```
git add .
git commit -m "added dockerfile"
git push -u origin master
```

Since we have not registered an SSH key with GitLab, we will need to enter a `Username` and `Password` when prompted.

In the browser open Drone at http://192.168.0.11:8000/root/helloworld-web and click on the build being executed to monitor progress.

Output for `build` resembles

```
+ make lint
go get github.com/golang/lint/golint
golint
+ make test
go test -v ./...
=== RUN   TestHandler
--- PASS: TestHandler (0.00s)
PASS
ok      _/drone/src/192.168.0.11/root/helloworld-web    0.027s
+ make build
go build -o helloworld-web -v
_/drone/src/192.168.0.11/root/helloworld-web
```

Mirroring what you saw in development in your local environment.

8.10.14. Author the publish step for the pipeline

Add the publish step to your `.drone.yml` so that the publish to the private registry will be automated through the pipeline. The `publish:` step must be indented the same as the prior `build:` step.

```
  publish:
    image: plugins/docker
    storage_driver: overlay
    insecure: true
    registry: 192.168.0.11:5000
    repo: 192.168.0.11:5000/nemonik/helloworld-web
    force_tag: true
    tags: [ latest ]
    custom_dns: ["128.29.154.114", "129.83.20.114"]
    when:
      branch: master
```

This step must be past the DNS servers otherwise the build will fail to resolve the corporate proxy when it attempts to retrieve external dependency to include in the container image this step builds

Push your code into GitLab

```
git add .
git commit -m "added publish step"
git push -u origin master
```

The publish step takes quite a bit of time and resebles:

```
+ /usr/local/bin/dockerd -g /var/lib/docker -s overlay --insecure-registry 192.168.0.11:5000 --dns 128.29.154.114 --dns 1
Registry credentials not provided. Guest mode enabled.
+ /usr/local/bin/docker version
Client:
 Version:       17.12.0-ce
 API version:   1.35
 Go version:    go1.9.2
 Git commit:    c97c6d6
 Built: Wed Dec 27 20:05:38 2017
 OS/Arch:       linux/amd64

Server:
 Engine:
  Version:      17.12.0-ce
  API version:  1.35 (minimum version 1.12)
  Go version:   go1.9.2
  Git commit:   c97c6d6
  Built:        Wed Dec 27 20:12:29 2017
  OS/Arch:      linux/amd64
  Experimental: false
+ /usr/local/bin/docker info
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 17.12.0-ce
Storage Driver: overlay
 Backing Filesystem: xfs
 Supports d_type: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 89623f28b87a6004d4b785663257362d1658a729
runc version: b2567b37d7b75eb4cf325b77297b140ea686ce8f
init version: 949e6fa
Security Options:
 seccomp
  Profile: default
Kernel Version: 3.10.0-693.21.1.el7.x86_64
Operating System: Alpine Linux v3.7 (containerized)
OSType: linux
Architecture: x86_64
CPUs: 4
Total Memory: 3.702GiB
Name: bb41b5849706
ID: 2CV5:2RY2:ID5N:XAAV:6VE7:NLCH:QC5I:AH6Z:RNYD:6HDD:G56W:SGMP
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
```

```
Debug Mode (server): false
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
 192.168.0.11:5000
 127.0.0.0/8
Live Restore Enabled: false

+ /usr/local/bin/docker build --rm=true -f Dockerfile -t ff06b0d6f51dc80d6ec4cb11dce59503932a9f49 . --pull=true --label o
Sending build context to Docker daemon  9.328MB

Step 1/9 : FROM 192.168.0.11:5000/nemonik/golang:1.10.1
1.10.1: Pulling from nemonik/golang
c73ab1c6897b: Pulling fs layer
1ab373b3deae: Pulling fs layer
b542772b4177: Pulling fs layer
57c8de432dbe: Pulling fs layer
c81227e1ec90: Pulling fs layer
df4d04d0f54c: Pulling fs layer
8ed488a16a31: Pulling fs layer
0828adce2551: Pulling fs layer
9b8c7e5da5a1: Pulling fs layer
0b398c7e2292: Pulling fs layer
30e8e73c0c04: Pulling fs layer
1de129fb171b: Pulling fs layer
27eab3e938d4: Pulling fs layer
17d8f862a4a2: Pulling fs layer
38a4faa53d09: Pulling fs layer
c81227e1ec90: Waiting
df4d04d0f54c: Waiting
30e8e73c0c04: Waiting
1de129fb171b: Waiting
8ed488a16a31: Waiting
0828adce2551: Waiting
27eab3e938d4: Waiting
17d8f862a4a2: Waiting
9b8c7e5da5a1: Waiting
38a4faa53d09: Waiting
0b398c7e2292: Waiting
57c8de432dbe: Waiting
b542772b4177: Verifying Checksum
b542772b4177: Download complete
1ab373b3deae: Verifying Checksum
1ab373b3deae: Download complete
57c8de432dbe: Verifying Checksum
57c8de432dbe: Download complete
c73ab1c6897b: Verifying Checksum
c73ab1c6897b: Download complete
8ed488a16a31: Verifying Checksum
8ed488a16a31: Download complete
c81227e1ec90: Verifying Checksum
c81227e1ec90: Download complete
9b8c7e5da5a1: Verifying Checksum
0828adce2551: Verifying Checksum
9b8c7e5da5a1: Download complete
0828adce2551: Download complete
30e8e73c0c04: Verifying Checksum
30e8e73c0c04: Download complete
```

```
0b398c7e2292: Verifying Checksum
0b398c7e2292: Download complete
27eab3e938d4: Verifying Checksum
27eab3e938d4: Download complete
17d8f862a4a2: Verifying Checksum
17d8f862a4a2: Download complete
38a4faa53d09: Verifying Checksum
38a4faa53d09: Download complete
1de129fb171b: Verifying Checksum
1de129fb171b: Download complete
df4d04d0f54c: Verifying Checksum
df4d04d0f54c: Download complete
c73ab1c6897b: Pull complete
1ab373b3deae: Pull complete
b542772b4177: Pull complete
57c8de432dbe: Pull complete
c81227e1ec90: Pull complete
df4d04d0f54c: Pull complete
8ed488a16a31: Pull complete
0828adce2551: Pull complete
9b8c7e5da5a1: Pull complete
0b398c7e2292: Pull complete
30e8e73c0c04: Pull complete
1de129fb171b: Pull complete
27eab3e938d4: Pull complete
17d8f862a4a2: Pull complete
38a4faa53d09: Pull complete
Digest: sha256:80864483daf76493863de6388a0bee977ddbc002d7b0cb19f253b3a1ddc8e14b
Status: Downloaded newer image for 192.168.0.11:5000/nemonik/golang:1.10.1
 ---> 7c239dc2f416
Step 2/9 : MAINTAINER Michael Joseph Walsh <nemonik@gmail.com>
 ---> Running in ccb7b3f16069
Removing intermediate container ccb7b3f16069
 ---> 674ade232277
Step 3/9 : RUN mkdir /app
 ---> Running in 7c1e60ab294d
Removing intermediate container 7c1e60ab294d
 ---> 695af24c4ba3
Step 4/9 : ADD main.go /app/
 ---> 935159a71682
Step 5/9 : WORKDIR /app
Removing intermediate container 49159cfd694e
 ---> 598bd28a1dcc
Step 6/9 : RUN go build -o helloworld-web .
 ---> Running in 9c7db7f708f9
Removing intermediate container 9c7db7f708f9
 ---> 161844c9f380
Step 7/9 : CMD ["/app/helloworld-web"]
 ---> Running in 4b2ae946a10c
Removing intermediate container 4b2ae946a10c
 ---> 379287345efa
Step 8/9 : EXPOSE 3000
 ---> Running in 8cfb85a26d01
Removing intermediate container 8cfb85a26d01
 ---> d3e4d8f8f384
Step 9/9 : LABEL "org.label-schema.build-date"='2018-04-18T20:23:07Z' "org.label-schema.vcs-ref"='ff06b0d6f51dc80d6ec4cb1
 ---> Running in 559bf2111f9b
Removing intermediate container 559bf2111f9b
 ---> 398456bad9b0
```

```
Successfully built 398456bad9b0
Successfully tagged ff06b0d6f51dc80d6ec4cb11dce59503932a9f49:latest
+ /usr/local/bin/docker tag ff06b0d6f51dc80d6ec4cb11dce59503932a9f49 192.168.0.11:5000/nemonik/helloworld-web:latest
+ /usr/local/bin/docker push 192.168.0.11:5000/nemonik/helloworld-web:latest
The push refers to repository [192.168.0.11:5000/nemonik/helloworld-web]
185786c47e18: Preparing
ca386051012f: Preparing
83a7149730ed: Preparing
5f69028652b8: Preparing
c6d8b0b39323: Preparing
a77a1a834917: Preparing
92ce06e068ca: Preparing
ab8d7a56d156: Preparing
30af257ded3b: Preparing
120f2e939f3b: Preparing
47823918137b: Preparing
2bd1ea5f0540: Preparing
f2ab791795ee: Preparing
e33f0ab4ff77: Preparing
20c527f217db: Preparing
61c06e07759a: Preparing
bcbe43405751: Preparing
e1df5dc88d2c: Preparing
120f2e939f3b: Waiting
47823918137b: Waiting
2bd1ea5f0540: Waiting
61c06e07759a: Waiting
f2ab791795ee: Waiting
e33f0ab4ff77: Waiting
20c527f217db: Waiting
a77a1a834917: Waiting
92ce06e068ca: Waiting
bcbe43405751: Waiting
ab8d7a56d156: Waiting
e1df5dc88d2c: Waiting
30af257ded3b: Waiting
5f69028652b8: Layer already exists
c6d8b0b39323: Layer already exists
a77a1a834917: Layer already exists
92ce06e068ca: Layer already exists
30af257ded3b: Layer already exists
ab8d7a56d156: Layer already exists
120f2e939f3b: Layer already exists
47823918137b: Layer already exists
2bd1ea5f0540: Layer already exists
f2ab791795ee: Layer already exists
e33f0ab4ff77: Layer already exists
20c527f217db: Layer already exists
61c06e07759a: Layer already exists
bcbe43405751: Layer already exists
ca386051012f: Pushed
83a7149730ed: Pushed
e1df5dc88d2c: Layer already exists
185786c47e18: Pushed
latest: digest: sha256:7f3030c35497e19a5677458be9b541f9e337c491d680a0fa5ad4e10127d50908 size: 4101
+ /usr/local/bin/docker rmi ff06b0d6f51dc80d6ec4cb11dce59503932a9f49
Untagged: ff06b0d6f51dc80d6ec4cb11dce59503932a9f49:latest
+ /usr/local/bin/docker system prune -f
Total reclaimed space: 0B
```

Indicating the `publish` executed successfully.

8.10.15. Add a deploy step to the pipeliine

Usually one adds a Drone secret and the `appleboy/ssh` plugin, but there appears to be a bug in the plugin, but one can accomplish the same goal using the plugin slightly differently.

First, enable the `Trusted` setting for the repository by opening

http://192.168.0.11:8000/root/helloworld-web/settings

And check off the following

Project Settings

- Trusted

You can leave the others to their defaults.

Look for Drone to float a modal at the bottom left denoting

```
Successfully updated the repository settings
```

indicating success.

Since we let Vagrant manage the creation and configuration of each Vagrant's SSH keys for each of our variants (e.g., dev, toolchain) vagrant will have to create a unique key pair for each vagrant. Vagrant store each vagrant's private inside the `.vagrant` path created in the class project.

The `toolchain` vagrant's private keys exists at

```
/vagrant/.vagrant/machines/toolchain/virtualbox/private_key
```

We can access each private key though from inside each vagrant, because we configured each to mount the class project to the path `/vagrant` with the line

```
    dev.vm.synced_folder ".", "/vagrant", type: "virtualbox"
```

placed inside of each vagrant's `config.vm.define` block.

For example, while on the `dev` vagrant enter the following command

```
ls /vagrant
```

It will show you class project.

Open `root/helloworld-web` 's `.drone.yml` in your editor and add an additional `deploy:` step below the ones you already have with the `deploy:` being indented the same as the prior `build:` and `deploy:` steps.

```
  deploy:
    image: appleboy/drone-ssh
    host: 192.168.0.11
    port: 22
    username: vagrant
    volumes:
      - /vagrant/.vagrant/machines/toolchain/virtualbox/private_key:/root/ssh/drone_rsa
    key_path: /root/ssh/drone_rsa
    command_timeout: 360
    script:
      - docker stop helloworld-web 2>/dev/null
      - docker rm helloworld-web 2>/dev/null
      - docker rmi 192.168.0.11:5000/nemonik/helloworld-web 2>/dev/null
      - docker run -d --restart=always --name helloworld-web --publish 3000:3000 192.168.0.11:5000/nemo
    when:
      branch: master
```

Commit the code to GitLab hosted remote repo

```
git add .drone.yml
git commit -m "added deploy step to pipeline"
git push -u origin master
```

Open on your host

http://192.168.0.11:8000/root/helloworld-web

And monitor the progress of the build. The pipeline should execute in a few minutes.

Once completed sucessfully, on your host open secure shell to your `toolchain` vagrant via

```
vagrant ssh toolchain
```

Then list the running docker containers filtering for the one the pipeline just created

```
docker ps --filter "name=helloworld-web"
```

Command line output will resemble

```
[root@toolchain drone]# docker ps --filter "name=helloworld-web"
CONTAINER ID        IMAGE                                        COMMAND                  CREATED             STATUS
9d08909cc2ee        192.168.0.11:5000/nemonik/helloworld-web   "/app/helloworld-web"   6 seconds ago       Up 5 seconds
```

Then on your docker host open

http://192.168.0.11:3000

The browser will render

```
Hello world!
```

So, now we have beggings of a real CI/CD pipeline. There are no strings on me.

8.10.16. Add to the pipeline SonarQube

So, let us add `sonar-scan` to our pipeline to update SonarQube with code quality scans automatically by opening `.drone.yml` in our text editor and adding the following step right after our `build` step, so that we run the step prior to a deploy.

```
  sonarqube:
    image: 192.168.0.11:5000/nemonik/golang-sonarqube-scanner:3.1.0.1141
    commands:
      - export DRONESRC=`pwd`
      - export GOBIN=$GOPATH/bin
      - mkdir -p $GOPATH/src/github.com/nemonik
      - cd $GOPATH/src/github.com/nemonik
      - ln -s $DRONESRC helloworld-web
      - gometalinter --install
      - gometalinter --deadline=120s --checkstyle > report.xml || true
      - gocov test ./... | gocov-xml || true
      - go test  -v ./... | go-junit-report > test.xml || true
      - sonar-scanner -X -D http.nonProxyHosts=$NO_PROXY -D http.proxyHost=$HTTP_PROXY -D sonar.host.ur
```

Things to note in the above

- This step uses an image building ontop of the 192.168.0.11:5000/nemonik/golang:1.10.1 image adding all the GoLang lint tools and the installation of `sonar-scanner` to speed builds along.
- And since, Drone mounts the source in the container in path other than /go/src/github.com/nemonik/helloworld-web` this is handled as well by this sub-section
- Cut-and-pasting may split the last `-sonar-scanner` line into multiple line resulting in your build failing. Correct in the editor.

```
      - export DRONESRC=`pwd`
      - export GOBIN=$GOPATH/bin
      - mkdir -p $GOPATH/src/github.com/nemonik
      - cd $GOPATH/src/github.com/nemonik
      - ln -s $DRONESRC helloworld-web
```

What follows handles running the scan absorbing errors as they arise, so as to not break the build.

```
      - gometalinter --install
      - gometalinter --deadline=120s --checkstyle > report.xml || true
      - gocov test ./... | gocov-xml || true
      - go test  -v ./... | go-junit-report > test.xml || true
      - sonar-scanner -X -D http.nonProxyHosts=$NO_PROXY -D http.proxyHost=$HTTP_PROXY -D sonar.host.ur
```

Commit the code to GitLab hosted remote repo

```
git add .drone.yml
git commit -m "added deploy step to pipeline"
git push -u origin master
```

Open on your host

http://192.168.0.11:8000/root/helloworld-web

And monitor the progress of the build. The pipeline should execute in a few minutes.

Typical success resembles

```
+ export DRONESRC=/drone/src/192.168.0.11/root/helloworld-web
+ export GOBIN=$GOPATH/bin
+ mkdir -p $GOPATH/src/github.com/nemonik
+ cd $GOPATH/src/github.com/nemonik
+ ln -s $DRONESRC helloworld-web
+ gometalinter --install
Installing:
  deadcode
  dupl
  errcheck
  gas
  goconst
  gocyclo
  goimports
  golint
  gosimple
  gotype
  gotypex
  ineffassign
  interfacer
  lll
  maligned
  megacheck
  misspell
  nakedret
  safesql
  staticcheck
  structcheck
  unconvert
  unparam
  unused
  varcheck
  vet
+ gometalinter --deadline=120s --checkstyle > report.xml || true
+ gocov test ./... | gocov-xml || true
warning: "./..." matched no packages
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE coverage SYSTEM "http://cobertura.sourceforge.net/xml/coverage-03.dtd">
<coverage line-rate="0" branch-rate="0" version="" timestamp="1524084561079">
        <packages></packages>
</coverage>
+ go test  -v ./... | go-junit-report > test.xml || true
warning: "./..." matched no packages
no packages to test
+ sonar-scanner -X -D http.nonProxyHosts=$NO_PROXY -D http.proxyHost=$HTTP_PROXY -D sonar.host.url=http://192.168.0.11:90
20:49:21.653 INFO: Scanner configuration file: /usr/local/sonar-scanner-3.1.0.1141-linux/conf/sonar-scanner.properties
20:49:21.661 INFO: Project root configuration file: NONE
20:49:21.712 INFO: SonarQube Scanner 3.1.0.1141
20:49:21.713 INFO: Java 1.8.0_121 Oracle Corporation (64-bit)
20:49:21.713 INFO: Linux 3.10.0-693.21.1.el7.x86_64 amd64
20:49:21.931 DEBUG: keyStore is :
20:49:21.931 DEBUG: keyStore type is : jks
20:49:21.931 DEBUG: keyStore provider is :
20:49:21.931 DEBUG: init keystore
20:49:21.932 DEBUG: init keymanager of type SunX509
20:49:22.029 DEBUG: Create: /root/.sonar/cache
20:49:22.030 INFO: User cache: /root/.sonar/cache
20:49:22.030 DEBUG: Create: /root/.sonar/cache/_tmp
```

```
20:49:22.033 DEBUG: Extract sonar-scanner-api-batch in temp...
20:49:22.045 DEBUG: Get bootstrap index...
20:49:22.045 DEBUG: Download: http://192.168.0.11:9000/batch/index
20:49:22.437 DEBUG: Get bootstrap completed
20:49:22.456 DEBUG: Download http://192.168.0.11:9000/batch/file?name=sonar-scanner-engine-shaded-7.0.jar to /root/.sonar
20:49:22.945 DEBUG: Create isolated classloader...
20:49:22.964 DEBUG: Start temp cleaning...
20:49:22.971 DEBUG: Temp cleaning done
20:49:22.972 DEBUG: Execution getVersion
20:49:22.975 INFO: SonarQube server 7.0.0
20:49:22.975 INFO: Default locale: "en_US", source code encoding: "US-ASCII" (analysis is platform dependent)
20:49:22.976 DEBUG: Work directory: /go/src/github.com/nemonik/.scannerwork
20:49:22.977 DEBUG: Execution execute
20:49:23.325 INFO: Publish mode
20:49:23.457 INFO: Load global settings
20:49:23.798 DEBUG: GET 200 http://192.168.0.11:9000/api/settings/values.protobuf | time=333ms
20:49:23.818 INFO: Load global settings (done) | time=362ms
20:49:23.830 INFO: Server id: AWLZ3KmeIXzDnCPHcQJq
20:49:23.877 INFO: User cache: /root/.sonar/cache
20:49:24.093 INFO: Load plugins index
20:49:24.169 DEBUG: GET 200 http://192.168.0.11:9000/api/plugins/installed | time=75ms
20:49:24.207 INFO: Load plugins index (done) | time=114ms
20:49:24.207 INFO: Load/download plugins
20:49:24.208 DEBUG: Download plugin 'sonar-golang-plugin-1.2.11-rc13.jar' to '/root/.sonar/cache/_tmp/fileCache4373472418
20:49:24.240 DEBUG: GET 200 http://192.168.0.11:9000/deploy/plugins/golang/sonar-golang-plugin-1.2.11-rc13.jar | time=30m
20:49:24.402 INFO: Load/download plugins (done) | time=195ms
20:49:24.550 DEBUG: Plugins:
20:49:24.550 DEBUG:   * Golang 1.2.11-rc13 (golang)
20:49:24.881 INFO: Process project properties
20:49:24.890 DEBUG: Process project properties (done) | time=8ms
20:49:24.910 INFO: Load project repositories
20:49:25.158 DEBUG: GET 200 http://192.168.0.11:9000/batch/project.protobuf?key=helloworld-web | time=247ms
20:49:25.215 INFO: Load project repositories (done) | time=305ms
20:49:25.286 DEBUG: Available languages:
20:49:25.286 DEBUG:   * GO => "go"
20:49:25.305 INFO: Load quality profiles
20:49:25.452 DEBUG: GET 200 http://192.168.0.11:9000/api/qualityprofiles/search.protobuf?projectKey=helloworld-web | time
20:49:25.462 INFO: Load quality profiles (done) | time=157ms
20:49:25.481 INFO: Load active rules
20:49:26.617 DEBUG: GET 200 http://192.168.0.11:9000/api/rules/search.protobuf?f=repo,name,severity,lang,internalKey,temp
20:49:26.669 INFO: Load active rules (done) | time=1188ms
20:49:26.675 INFO: Load metrics repository
20:49:26.754 DEBUG: GET 200 http://192.168.0.11:9000/api/metrics/search?f=name,description,direction,qualitative,custom&p
20:49:26.769 INFO: Load metrics repository (done) | time=94ms
20:49:26.779 WARN: SCM provider autodetection failed. No SCM provider claims to support this project. Please use sonar.sc
20:49:26.788 INFO: Project key: helloworld-web
20:49:26.789 DEBUG: Start recursive analysis of project modules
20:49:26.790 INFO: -------------  Scan helloworld-web
20:49:26.854 INFO: Load server rules
20:49:26.884 DEBUG: GET 200 http://192.168.0.11:9000/api/rules/list.protobuf | time=29ms
20:49:26.889 INFO: Load server rules (done) | time=35ms
20:49:26.918 INFO: Base dir: /go/src/github.com/nemonik
20:49:26.918 INFO: Working dir: /go/src/github.com/nemonik/.scannerwork
20:49:26.920 INFO: Source paths: .
20:49:26.920 INFO: Source encoding: US-ASCII, default locale: en_US
20:49:26.946 DEBUG: Declared extensions of language GO were converted to sonar.lang.patterns.go : **/*.go
20:49:26.949 DEBUG: Initializers :
20:49:26.950 INFO: Index files
20:49:26.958 INFO: Excluded sources:
```

```
20:49:26.958 INFO:   **/*test.go
20:49:26.973 DEBUG: 'helloworld-web/Dockerfile' indexed with language 'null'
20:49:26.975 DEBUG: 'helloworld-web/main.go' indexed with language 'go'
20:49:26.975 DEBUG: 'helloworld-web/Makefile' indexed with language 'null'
20:49:26.975 DEBUG: 'helloworld-web/helloworld-web' indexed with language 'null'
20:49:26.975 DEBUG: 'report.xml' indexed with language 'null'
20:49:26.975 DEBUG: 'test.xml' indexed with language 'null'
20:49:26.977 INFO: 6 files indexed
20:49:26.977 INFO: 1 file ignored because of inclusion/exclusion patterns
20:49:26.978 INFO: Quality profile for go: Golint Rules
20:49:26.993 DEBUG: 'Generic Coverage Report' skipped because one of the required properties is missing
20:49:26.994 DEBUG: 'Generic Test Executions Report' skipped because one of the required properties is missing
20:49:27.008 DEBUG: 'Generic Coverage Report' skipped because one of the required properties is missing
20:49:27.008 DEBUG: 'Generic Test Executions Report' skipped because one of the required properties is missing
20:49:27.014 DEBUG: Sensors : GoMetaLinter issues loader sensor -> Go Coverage -> Go test JUnit loader sensor -> Go Highl
20:49:27.014 INFO: Sensor GoMetaLinter issues loader sensor [golang]
20:49:27.028 INFO: Parsing the file report.xml
20:49:27.028 INFO: Parsing 'GoMetaLinter' Analysis Results
20:49:27.029 DEBUG: Parsing file /go/src/github.com/nemonik/report.xml
20:49:27.101 INFO: Sensor GoMetaLinter issues loader sensor [golang] (done) | time=87ms
20:49:27.101 INFO: Sensor Go Coverage [golang]
20:49:27.101 INFO: /go/src/github.com/nemonik
20:49:27.106 DEBUG: Path in stream/go/src/github.com/nemonik
20:49:27.106 INFO: no coverage file in package /go/src/github.com/nemonik/coverage.xml
20:49:27.106 DEBUG: Path in stream/go/src/github.com/nemonik/helloworld-web
20:49:27.106 INFO: no coverage file in package /go/src/github.com/nemonik/helloworld-web/coverage.xml
20:49:27.107 DEBUG: Path in stream/go/src/github.com/nemonik/report.xml
20:49:27.107 DEBUG: Path in stream/go/src/github.com/nemonik/test.xml
20:49:27.132 DEBUG: 'helloworld-web/main.go' generated metadata  with charset 'US-ASCII'
20:49:27.147 INFO: Sensor Go Coverage [golang] (done) | time=46ms
20:49:27.147 INFO: Sensor Go test JUnit loader sensor [golang]
20:49:27.149 INFO: base dir /go/src/github.com/nemonik
20:49:27.150 DEBUG: {}
20:49:27.150 DEBUG: {}
20:49:27.153 INFO: Sensor Go test JUnit loader sensor [golang] (done) | time=6ms
20:49:27.153 INFO: Sensor Go Highlighter Sensor [golang]
20:49:27.156 DEBUG: Coloring the file: /go/src/github.com/nemonik/helloworld-web/main.go
20:49:27.167 DEBUG: Line number 1 index start: 0 index end: 7
20:49:27.169 DEBUG: Line number 3 index start: 0 index end: 6
20:49:27.169 DEBUG:     "fmt" 1
20:49:27.169 DEBUG:     "fmt" 5
20:49:27.169 DEBUG: index 1 indexEnd 5
20:49:27.169 DEBUG:     "fmt" -1
20:49:27.169 DEBUG:     "fmt" 1
20:49:27.169 DEBUG:     "net/http" 1
20:49:27.169 DEBUG:     "net/http" 10
20:49:27.170 DEBUG: index 1 indexEnd 10
20:49:27.170 DEBUG:     "net/http" -1
20:49:27.170 DEBUG:     "net/http" 1
20:49:27.170 DEBUG: Line number 8 index start: 0 index end: 4
20:49:27.170 DEBUG:     http.HandleFunc("/", handler) 17
20:49:27.170 DEBUG:     http.HandleFunc("/", handler) 19
20:49:27.170 DEBUG: index 17 indexEnd 19
20:49:27.171 DEBUG:     http.HandleFunc("/", handler) -1
20:49:27.171 DEBUG:     http.HandleFunc("/", handler) 17
20:49:27.171 DEBUG:     fmt.Print("listening on :3000\n") 11
20:49:27.171 DEBUG:     fmt.Print("listening on :3000\n") 32
20:49:27.171 DEBUG: index 11 indexEnd 32
20:49:27.171 DEBUG:     fmt.Print("listening on :3000\n") -1
```

```
20:49:27.171 DEBUG:     fmt.Print("listening on :3000\n") 11
20:49:27.171 DEBUG: Line number 10 index start: 7 index end: 10
20:49:27.172 DEBUG:     http.ListenAndServe(":3000", logRequest(http.DefaultServeMux)) 21
20:49:27.172 DEBUG:     http.ListenAndServe(":3000", logRequest(http.DefaultServeMux)) 27
20:49:27.172 DEBUG: index 21 indexEnd 27
20:49:27.172 DEBUG:     http.ListenAndServe(":3000", logRequest(http.DefaultServeMux)) -1
20:49:27.172 DEBUG:     http.ListenAndServe(":3000", logRequest(http.DefaultServeMux)) 21
20:49:27.172 DEBUG: Line number 14 index start: 0 index end: 4
20:49:27.172 DEBUG:     fmt.Fprintf(w, "Hello world!") 16
20:49:27.172 DEBUG:     fmt.Fprintf(w, "Hello world!") 29
20:49:27.173 DEBUG: index 16 indexEnd 29
20:49:27.173 DEBUG:     fmt.Fprintf(w, "Hello world!") -1
20:49:27.173 DEBUG:     fmt.Fprintf(w, "Hello world!") 16
20:49:27.173 DEBUG: Line number 15 index start: 8 index end: 11
20:49:27.173 DEBUG: Line number 18 index start: 0 index end: 4
20:49:27.173 DEBUG: Line number 19 index start: 25 index end: 29
20:49:27.173 DEBUG: Line number 19 index start: 1 index end: 7
20:49:27.173 DEBUG:             fmt.Printf("%s  %s  %s\n", r.RemoteAddr, r.Method, r.URL) 13
20:49:27.173 DEBUG:             fmt.Printf("%s  %s  %s\n", r.RemoteAddr, r.Method, r.URL) 26
20:49:27.174 DEBUG: index 13 indexEnd 26
20:49:27.174 DEBUG:             fmt.Printf("%s  %s  %s\n", r.RemoteAddr, r.Method, r.URL) -1
20:49:27.174 DEBUG:             fmt.Printf("%s  %s  %s\n", r.RemoteAddr, r.Method, r.URL) 13
20:49:27.174 DEBUG: Line number 20 index start: 8 index end: 11
20:49:27.183 INFO: Sensor Go Highlighter Sensor [golang] (done) | time=30ms
20:49:27.183 INFO: Sensor Go Metrics Sensor [golang]
20:49:27.186 INFO: Sensor Go Metrics Sensor [golang] (done) | time=3ms
20:49:27.186 INFO: Sensor Zero Coverage Sensor
20:49:27.199 INFO: Sensor Zero Coverage Sensor (done) | time=13ms
20:49:27.199 INFO: Sensor CPD Block Indexer
20:49:27.200 DEBUG: org.sonar.scanner.cpd.deprecated.DefaultCpdBlockIndexer is used for go
20:49:27.200 DEBUG: No CpdMapping for language go
20:49:27.200 INFO: Sensor CPD Block Indexer (done) | time=1ms
20:49:27.200 INFO: No SCM system was detected. You can use the 'sonar.scm.provider' property to explicitly specify it.
20:49:27.201 INFO: Calculating CPD for 0 files
20:49:27.202 INFO: CPD calculation finished
20:49:27.292 INFO: Analysis report generated in 88ms, dir size=21 KB
20:49:27.302 INFO: Analysis reports compressed in 10ms, zip size=5 KB
20:49:27.303 INFO: Analysis report generated in /go/src/github.com/nemonik/.scannerwork/scanner-report
20:49:27.303 DEBUG: Upload report
20:49:27.410 DEBUG: POST 200 http://192.168.0.11:9000/api/ce/submit?projectKey=helloworld-web&projectName=helloworld-web
20:49:27.413 INFO: Analysis report uploaded in 109ms
20:49:27.414 INFO: ANALYSIS SUCCESSFUL, you can browse http://192.168.0.11:9000/dashboard/index/helloworld-web
20:49:27.414 INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted
20:49:27.414 INFO: More about the report processing at http://192.168.0.11:9000/api/ce/task?id=AWLahO0mE0IcvTpxuSn_
20:49:27.415 DEBUG: Report metadata written to /go/src/github.com/nemonik/.scannerwork/report-task.txt
20:49:27.417 DEBUG: Post-jobs :
20:49:27.422 INFO: Task total time: 2.869 s
20:49:27.505 INFO: ------------------------------------------------------------------------
20:49:27.505 INFO: EXECUTION SUCCESS
20:49:27.505 INFO: ------------------------------------------------------------------------
20:49:27.505 INFO: Total time: 5.914s
20:49:27.550 INFO: Final Memory: 6M/129M
20:49:27.550 INFO: ------------------------------------------------------------------------
```

8.10.17. Author an automated funtional test

We're going to write an automated functional test of `helloworld-web` instead of relying on a manual functional test by using Selenium, a portable software-testing framework for web applications. Essentially, Selenium automates web browsers.

More can be found here

http://www.seleniumhq.org/

You'll need three shells open to your `dev` vagrant to complete this section.

8.10.17.1. Run the *helloworld-web* application

In the first `vagrant ssh dev` enter

```
docker run -p 3000:3000 nemonik/helloworld-web
```

We've seen the log out before, but it looks like

```
[vagrant@dev ~]$ docker run -p 3000:3000 nemonik/helloworld-web
listening on :3000
```

8.10.17.2. Pull and run Selenium Firefox Standalone

In the second `vagrant ssh dev` enter

```
docker pull 192.168.0.11:5000/nemonik/standalone-firefox:3.11
```

to pull the container image frome our private Docker repository running on our `toolchain` vagrant.

Then enter into the command line

```
docker run -p 4444:4444 192.168.0.11:5000/nemonik/standalone-firefox:3.11
```

This stands up Selenium specifically running Firefox. The container is running in the foreground so we can watch the log output.

A good start outputs to the command line

```
[vagrant@dev ~]$ docker run -p 4444:4444 192.168.0.11:5000/nemonik/standalone-firefox:3.11
20:58:41.002 INFO [GridLauncherV3.launch] - Selenium build info: version: '3.11.0', revision: 'e59cfb3'
20:58:41.003 INFO [GridLauncherV3$1.launch] - Launching a standalone Selenium Server on port 4444
2018-04-18 20:58:41.093:INFO::main: Logging initialized @335ms to org.seleniumhq.jetty9.util.log.StdErrLog
20:58:41.265 INFO [SeleniumServer.boot] - Welcome to Selenium for Workgroups....
20:58:41.266 INFO [SeleniumServer.boot] - Selenium Server is up and running on port 4444
```

8.10.17.3. Create our test automation

IIn another `vagrant ssh` to `dev` , we author and run our automated test.

In this other command line at the root of the `helloworld-web` project (e.g., `/home/vagrant/go/src/github.com/nemonik/helloworld-web`) enter

```
mkdir selenium-test
cd selenium-test
```

We're going to write our test in Python. Python is already installed on the `dev` vagrant. Ansible uses it.

We'll need to install a dependency, we can install by entering into the command line

```
sudo pip install selenium
```

But let's create a `requirements.txt` file to hold our dependency with our text editor containing

```
selenium==3.11
```

This results in allow the test's dependencies being enumerated in a file including the specififc version of the dependency, so their installation can be automated.

Then test out the requirements.txt file by entering

```
sudo pip install -r requirements.txt
```

Then in another editor create our `test_hello_world.py` Python-based unit test containing

```
# Copyright (C) 2018 Michael Joseph Walsh - All Rights Reserved
# You may use, distribute and modify this code under the
# terms of the the license.
#
# You should have received a copy of the license with
# this file. If not, please email <mjwalsh@nemonik.com>

import unittest
from selenium import webdriver
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities
import sys
import os

class PythonOrgSearch(unittest.TestCase):

    def setUp(self):

        webdriver.DesiredCapabilities.FIREFOX['proxy'] = {
            "proxyType":"DIRECT"
#                "noProxy":"os.environ['no_proxy']," + selenium_host # Selenium cannot handle no_proxy
    }

        self.driver = webdriver.Remote(command_executor="http://" + selenium_host + ":4444/wd/hub", des

    def test_helloworld_web(self):
        driver = self.driver
        driver.get(helloworld_web_url)
        assert "Hello world!" in driver.page_source

    def tearDown(self):
        self.driver.close()

if __name__ == "__main__":
    if len(sys.argv) == 3:
        selenium_host = sys.argv[1]
        helloworld_web_url = sys.argv[2]
    else:
        print "python test_hello_world.py selenium_host helloworld_web_url"
        sys.exit()

    del sys.argv[1:]
    unittest.main()
```

Save the file and exit.

You might notice the commented out section about `Selenium cannot handle no_proxy.` Yeah. This will later present a problem.

For now, now let us run our test by entering into the command line

```
python test_hello_world.py  192.168.0.10 http://192.168.0.10:3000
```

Successful command line output in this window will be

```
[vagrant@dev selenium-test]$ python test_hello_world.py  192.168.0.10 http://192.168.0.10:3000
.
----------------------------------------------------------------------
Ran 1 test in 2.965s
```

The other windows will show logging.

`ctrl-c` to stop the containers in your other shells.

8.10.18. Add a *selenium* step to the pipeline

In one of your exsiting `vagrant ssh` to `dev` enter into the commmand line

```
cd /home/vagrant/go/src/github.com/nemonik/helloworld-web
```

Edit the `.drone.yml` file and add to the bottom the following with the `selenum:` step having the same idention as the prior steps, and `services:` starting the line.

```
  selenium:
    image: 192.168.0.11:5000/nemonik/python:2.7.14
    commands:
      - export NO_PROXY=$NO_PROXY,$(python selenium-test/resolve.py firefox)
      - export no_proxy=$no_proxy,$(python selenium-test/resolve.py firefox)
      - cd selenium-test
      - pip install -r requirements.txt
      - python test_hello_world.py firefox http://192.168.0.11:3000

services:
  firefox:
    image: 192.168.0.11:5000/nemonik/standalone-firefox:3.11
    volumes:
      - /dev/shm:/dev/shm
    ports:
      - "4444:4444"
```

Drone uses the `services:` section to spin up a patched version of `selenium/standalone-firefox:3.9.1` exposed with the name `firefox` . This is where the problem creeps in that I mentioned earlier, where the `selenium` step will hit corporate proxy to resolve `firefox` . We don't want that to happen. so the `selenium:` step will makes use of another Python script to resolved `firefox` service to its private IP and add it the `no_proxy` and `NO_PROXY` environmental variables, so that the Python selenium test code doesn't attempt to pass the request on to the corporate proxy and result in the test failing.

Create in a text editor `selenium-test/resolve.py` with

```
#!/usr/bin/env python

# Copyright (C) 2018 Michael Joseph Walsh - All Rights Reserved
# You may use, distribute and modify this code under the
# terms of the the license.
#
# You should have received a copy of the license with
# this file. If not, please email <mjwalsh@nemonik.com>

import socket, sys

if __name__ == "__main__":
    if len(sys.argv) == 2:
        print socket.gethostbyname( sys.argv[1])
```

Commit the code to GitLab hosted remote repo

```
git add .
git commit -m "added selenium step to pipeline"
git push -u origin master
```

Open on your host

http://192.168.0.11:8000/root/helloworld-web

And monitor the progress of the build. The pipeline should execute in a few minutes.

Successful output for this stage resembles

```
+ export NO_PROXY=$NO_PROXY,$(python selenium-test/resolve.py firefox)
+ export no_proxy=$no_proxy,$(python selenium-test/resolve.py firefox)
+ cd selenium-test
+ pip install -r requirements.txt
Collecting selenium==3.11 (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/5e/1f/6c2204b9ae14eddab615c5e2ee4956c65ed533e0a9986c23eabd801ae849/
Installing collected packages: selenium
Successfully installed selenium-3.11.0
+ python test_hello_world.py firefox http://192.168.0.11:3000
.
----------------------------------------------------------------------
Ran 1 test in 7.731s

OK
```

8.10.19. Add a DAST step (OWASP ZAP) to the pipeline

Dynamic application security testing (DAST) is used to detect security vulnerabilities in an application while it is running, so as to help you remediate these concerns while in development.

The OWASP Zed Attack Proxy (ZAP) is one of the world's most popular free DAST tools actively maintained by hundreds of international volunteers, so add a step to test the application.

First exercise ZAP locally on dev by entering in the command line in a `vagrant ssh` to `dev` at the root of the project

```
docker pull 192.168.0.11:5000/nemonik/zap2docker-stable:2.7.0
docker run 192.168.0.11:5000/nemonik/zap2docker-stable:2.7.0 zap-baseline.py -t http://192.168.0.11:3000
```

Success in the command line resembles

```
[vagrant@dev helloworld-web]$ docker pull 192.168.0.11:5000/nemonik/zap2docker-stable:2.7.0
2.7.0: Pulling from nemonik/zap2docker-stable
660c48dd555d: Pull complete
4c7380416e78: Pull complete
421e436b5f80: Pull complete
e4ce6c3651b3: Pull complete
be588e74bd34: Pull complete
2e09786d5b2f: Pull complete
197722f79cca: Pull complete
bd33ded2095e: Pull complete
4b06ea4ab95b: Pull complete
5549c936d696: Pull complete
c2bfc8c8addb: Pull complete
1a4b29516a99: Pull complete
8ce1a9f9c46c: Pull complete
8808d1ac4805: Pull complete
24eed6f4cce3: Pull complete
2cd87b08c44e: Pull complete
5ae7b6620997: Pull complete
d6d847cf31e3: Pull complete
e844d15f6458: Pull complete
3e0203eff2eb: Pull complete
67e08593d215: Pull complete
Digest: sha256:00e3d7d88cd27ae2b5d4639cae5a6aed517fa379cdfaf9fcd482c5f79d88b226
Status: Downloaded newer image for 192.168.0.11:5000/nemonik/zap2docker-stable:2.7.0
[vagrant@dev helloworld-web]$ docker run 192.168.0.11:5000/nemonik/zap2docker-stable:2.7.0 zap-baseline.py -t http://192.
Apr 18, 2018 9:15:11 PM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
Total of 3 URLs
PASS: Cookie No HttpOnly Flag [10010]
PASS: Cookie Without Secure Flag [10011]
PASS: Password Autocomplete in Browser [10012]
PASS: Incomplete or No Cache-control and Pragma HTTP Header Set [10015]
PASS: Web Browser XSS Protection Not Enabled [10016]
PASS: Cross-Domain JavaScript Source File Inclusion [10017]
PASS: Content-Type Header Missing [10019]
PASS: X-Frame-Options Header Scanner [10020]
PASS: Secure Pages Include Mixed Content [10040]
PASS: Private IP Disclosure [2]
PASS: Session ID in URL Rewrite [3]
PASS: Script Passive Scan Rules [50001]
PASS: Application Error Disclosure [90022]
WARN-NEW: X-Content-Type-Options Header Missing [10021] x 4
        http://192.168.0.11:3000/
        http://192.168.0.11:3000/robots.txt
        http://192.168.0.11:3000
        http://192.168.0.11:3000/sitemap.xml
FAIL-NEW: 0     FAIL-INPROG: 0  WARN-NEW: 1     WARN-INPROG: 0  INFO: 0 IGNORE: 0       PASS: 13
[vagrant@dev helloworld-web]$
[vagrant@dev helloworld-web]$
```

Great, now lets add a step to our pipeline to the same after the `selenium:`

```
  owasp-zaproxy:
    image: 192.168.0.11:5000/nemonik/zap2docker-stable:2.7.0
    commands:
      - zap-baseline.py -t http://192.168.0.11:3000 || true
```

Commit the code to GitLab hosted remote repo

```
git add .
git commit -m "added owasp-zaproxy step to pipeline"
git push -u origin master
```

Open on your host

http://192.168.0.11:8000/root/helloworld-web

And monitor the progress of the build. The pipeline should execute in a few minutes.

Successful output for this stage resembles

```
+ zap-baseline.py -t http://192.168.0.11:3000 || true
Apr 18, 2018 9:19:10 PM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
Total of 3 URLs
PASS: Cookie No HttpOnly Flag [10010]
PASS: Cookie Without Secure Flag [10011]
PASS: Password Autocomplete in Browser [10012]
PASS: Incomplete or No Cache-control and Pragma HTTP Header Set [10015]
PASS: Web Browser XSS Protection Not Enabled [10016]
PASS: Cross-Domain JavaScript Source File Inclusion [10017]
PASS: Content-Type Header Missing [10019]
PASS: X-Frame-Options Header Scanner [10020]
PASS: Secure Pages Include Mixed Content [10040]
PASS: Private IP Disclosure [2]
PASS: Session ID in URL Rewrite [3]
PASS: Script Passive Scan Rules [50001]
PASS: Application Error Disclosure [90022]
WARN-NEW: X-Content-Type-Options Header Missing [10021] x 4
        http://192.168.0.11:3000/
        http://192.168.0.11:3000/robots.txt
        http://192.168.0.11:3000/sitemap.xml
        http://192.168.0.11:3000
FAIL-NEW: 0     FAIL-INPROG: 0  WARN-NEW: 1     WARN-INPROG: 0  INFO: 0 IGNORE: 0       PASS: 13
```

The application is relatively simple, so it was doubtful anything would be found, but there is a warning that a *X-Content-Type-Options Header* is missing that could be resolved by adding addtional handlers to the helloworld-web's main.go.

8.10.20. All the source for *helloworld-web*

The `helloworld-web` project can be viewed completed at

https://gitlab.mitre.org/mjwalsh/helloworld-web

8.11. Using what you've learned

Git clone the Python+Flask Magic Eight Ball web app

https://gitlab.mitre.org/mjwalsh/magiceightball

and develop a pipeline for.

8.12. Shoo away your vagrants

If you're done with your vagrants, shoo them away from the root of the project on the host

```
vagrant destroy -f
```

And they're gone.

8.13. That's it

That's a wrap.