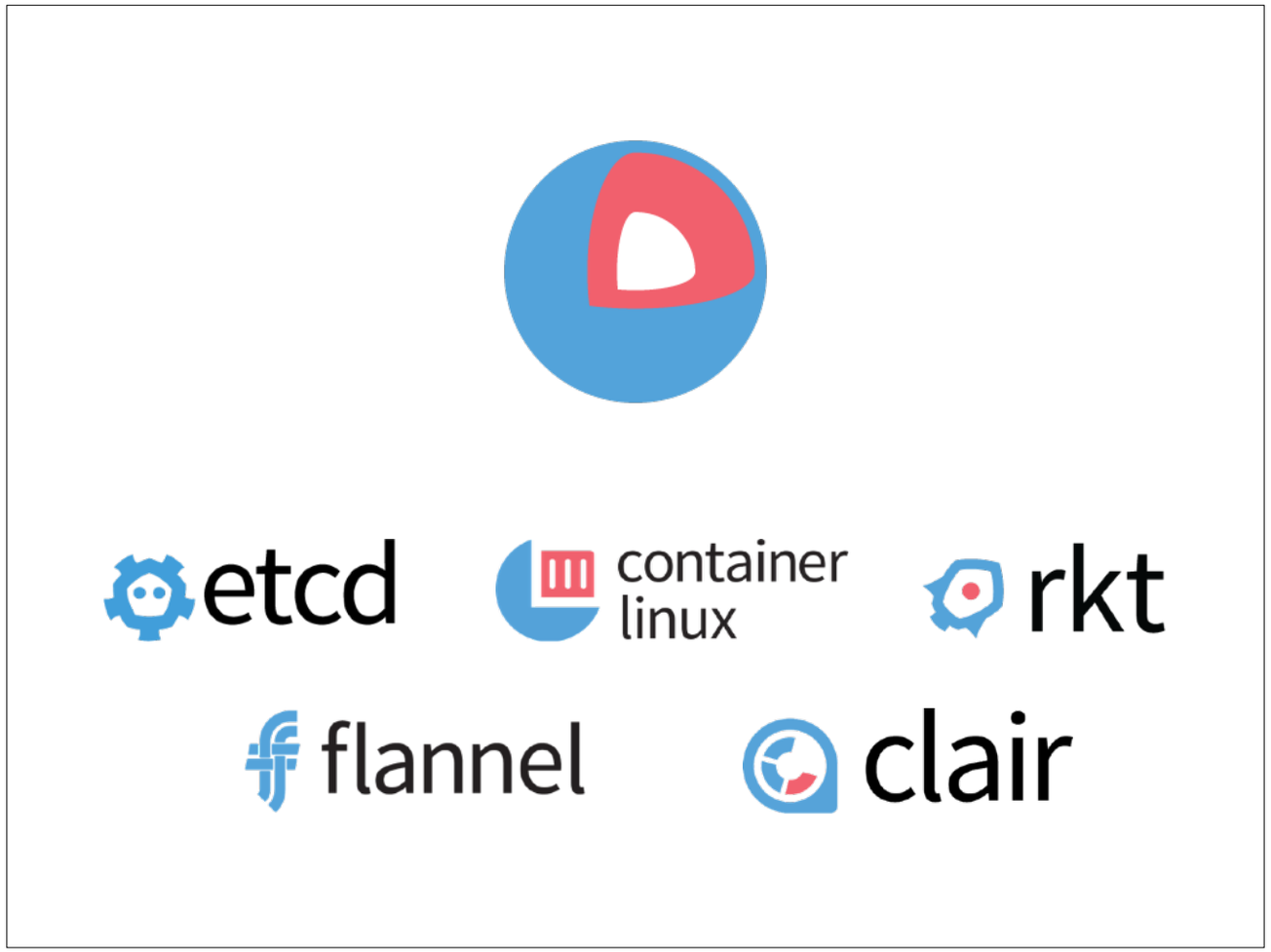
Kubernetes on Container Linux

My Experience

Slightly different talk from other talks as it’s not technical.

Ambrose Chua
@serverwentdown

I’m still learning…



These are the main open source projects by CoreOS. Most notably would be the key-value store etcd, used in Kubernetes, and flannel, an overlay network supported by Kubernetes. rkt is a container runtime that supports the open container image, Docker and the now-deprecated appc spec. Clair is a vulnerability analysis tool for containers.

Container Linux is my favourite project. It’s based on the Chromium OS project, and unlike a conventional distribution, all that is contained in Container Linux is the essentials to run containers: Docker, rkt and other CoreOS tools. No managing of packages.

Partition table

NUMBER	LABEL	DESCRIPTION	PARTITION TYPE
1	EFI-SYSTEM	Contains the bootloader	VFAT
2	BIOS-BOOT	This partition is reserved for future use	(none)
3	USR-A	One of two active/passive partitions holding Container Linux	EXT4
4	USR-B	One of two active/passive partitions holding Container Linux	(empty on first boot)
5	ROOT-C	This partition is reserved for future use	(none)
6	OEM	Stores configuration data specific to an OEM platform	EXT4
7	OEM-CONFIG	Optional storage for an OEM	(defined by OEM)
8	(unused)	This partition is reserved for future use	(none)
9	ROOT	Stateful partition for storing persistent data	EXT4, BTRFS, or XFS

For more information, read more about the disk layout used by Chromium and ChromeOS, which inspired the layout used by Container Linux.

A typical Linux distribution would have at least one root filesystem and multiple mounts. However Container Linux adopts a partition layout similar to ChromiumOS, where there are two /usr partitions that contain readonly data. These partitions contain the core of the operating system, and since there are two partitions you can overwrite the unmounted one with an update and reboot to take effect.

```
#!/bin/bash
echo -n "Did you forget to add DNS records? Press enter to continue"
read
u=$1
p=$(/hosting/tools/random-port.sh)
mkdir -p $u/www
echo "" > $u/access.log
echo "" > $u/error.log
cat ${0%/*}/nginx.conf.sample | sed "s/USER/$u/g" | sed "s/RAND_PORT/$p/g"
inx.conf
cat ${0%/*}/profile.sample | sed "s/USER/$u/g" | sed "s/RAND_PORT/$p/g" >
ile
mkdir -p $u/.config/systemd/user/default.target.wants
cat ${0%/*}/app.service.sample | sed "s/USER/$u/g" | sed "s/RAND_PORT/$p/g
config/systemd/user/app.service
loginctl enable-linger $u
cd $u
echo -n "Please restart nginx now, then press enter: "
read
set -e
cd ..
/hosting/tools/get-cert -w $PWD/$u/www -d $u.makerforce.io
set +e
@
NORMAL  /hosting/tools/create.sh                    unix < utf-8 < sh  80%
```

So, one day, I decided to try to get rid of my old server setup. It ran on a single node, uses configuration files and scripts to perform limited automation.

```
      - partials/
      - author.hbs
      - default.hbs
      - index.hbs
      - page.hbs
      - post.hbs
      - tag.hbs
      - package.json

deploy:
  stage: deploy
  script:
    - echo "Deploying... "
    - eval $(ssh-agent -s)
    - ssh-add <(echo "$GITLAB_DEPLOY_KEY")
    - mkdir -p ~/.ssh
    - scp -o StrictHostKeyChecking=no -r assets/ partials/ author.
    - ssh gitlab-deploy@node1.makerforce.io sudo -u hosting XDG_RU
  only:
    - master
```

There was no continuous integration except for SCPing files from GitLab CI to the host machine.

Running Kubernetes

And even though I had very limited experience with containers, I knew it was a good direction to go. So I decided on running Kubernetes on Container Linux, as a challenge for myself.

Step 1: Installing Container Linux

Container Linux can be booted many ways. I decided to install it to disk using the ISO, on bare metal. I chose bare metal because I didn’t want the overhead of a VM, and the need to manage it. I also would eventually migrate everything to Containers, even if the containers were stateful.

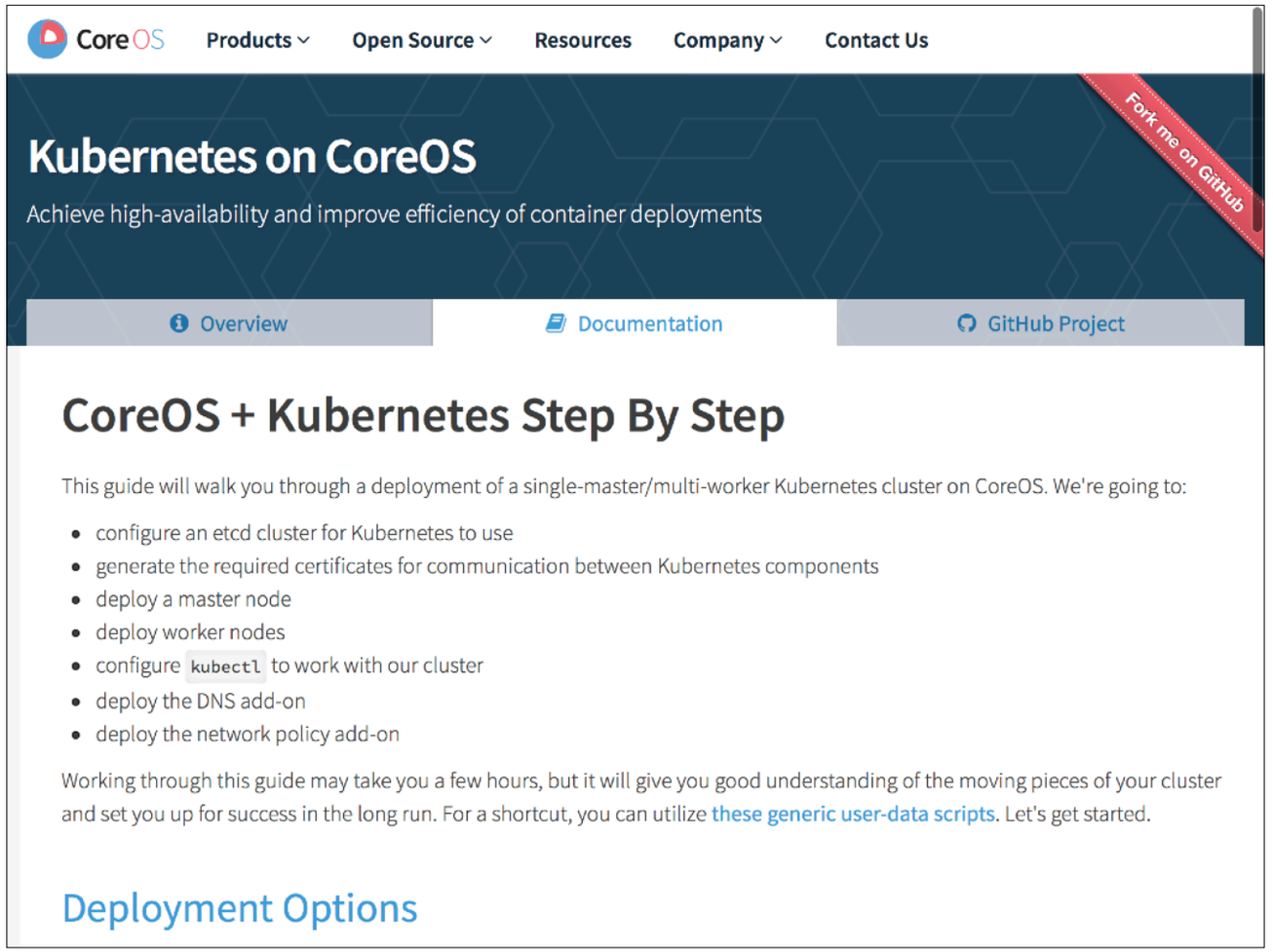
coreos-install -d /dev/sda -i ignition.json

Container Linux also recently introduced a change to their configuration through userdata. (Explain cloud-init) Container Linux has a YML-based configuration that transpires to JSON using a tool, and that can be put into userdata. It includes many CoreOS-specific configuration, and reduces the amount of configuration.

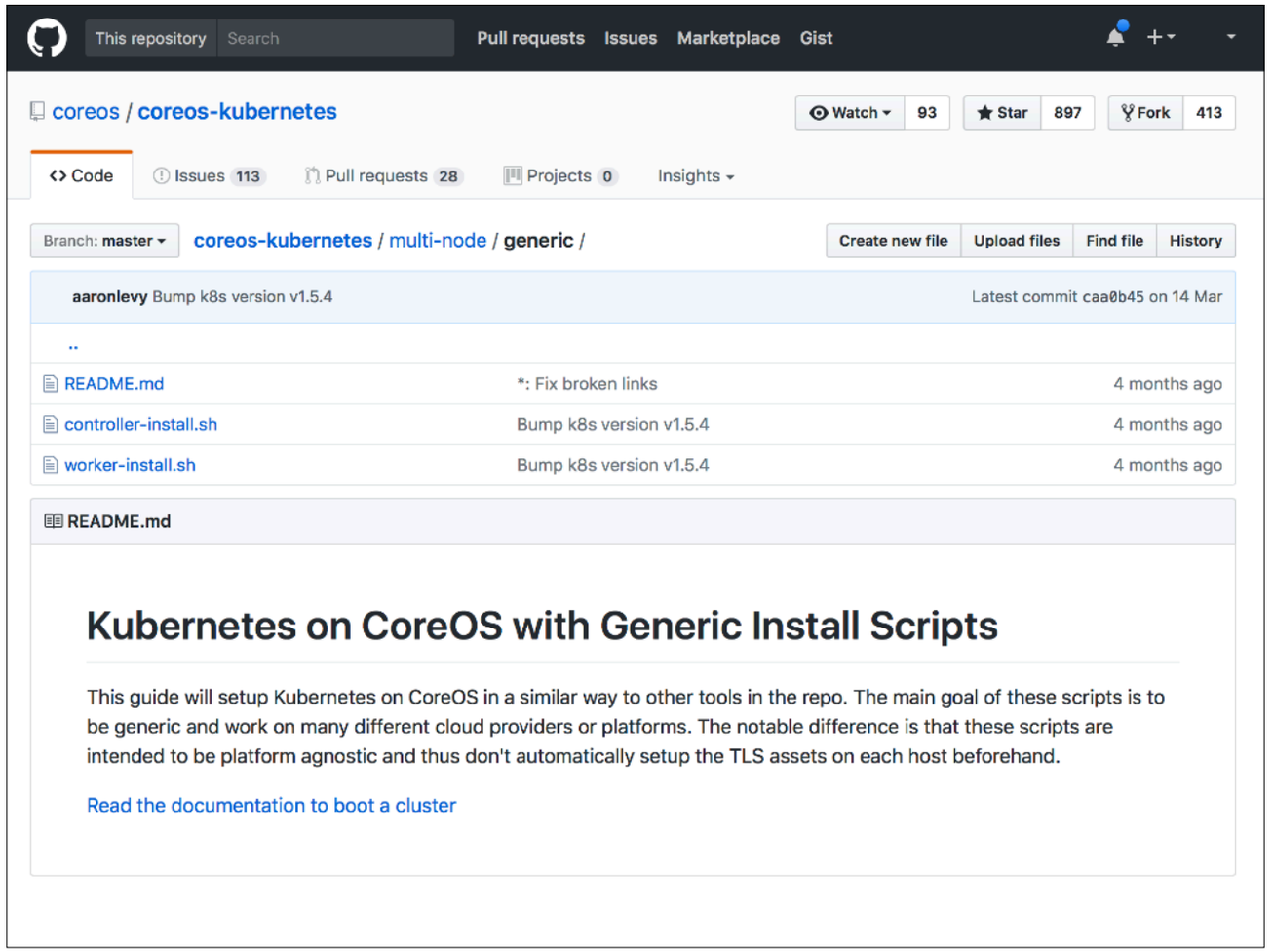
```
#cloud-config

passwd:
  users:
    - name: core
      ssh_authorized_keys:
        - “ecdsa-sha2-nistp256 …”

etcd:
  version: 3.1.6
  name: "controller"
  discovery: “https://discovery.etcd.io/…”
  listen_client_urls: “http://10.0.1.1:2379,http://127.0.0.1:2379”
  advertise_client_urls: "http://10.0.1.1:2379"
  listen_peer_urls: "http://10.0.1.1:2380"
  initial_advertise_peer_urls: "http://10.0.1.1:2380"

locksmith:
  reboot_strategy: "etcd-lock"

networkd:
  units:
    - name: static.network
      contents: |
        [Match]
        Name=enp2s0

        [Network]
        Address=10.0.1.1/22
        Gateway=10.0.0.1
        DNS=10.0.0.1

…
```

{"ignition":{"version":"2.0.0","config":{}},"storage":{"files":
[{"filesystem":"root","path":"/etc/hostname","contents":
{"source":"data:,controller","verification":{}},"mode":420,"user":
{},"group":{}},{"filesystem":"root","path":"/etc/coreos/
update.conf","contents":{"source":"data:,
%0AREBOOT_STRATEGY%3D%22etcd-
lock%22","verification":{}},"mode":420,"user":{},"group":
{}}]},"systemd":{"units":[{"name":"etcd-
member.service","enable":true,"dropins":[{"name":"20-clct-
etcd-member.conf","contents":"[Service]
\nEnvironment=\"ETCD_IMAGE_TAG=v3.1.6\"\nExecStart=\n
ExecStart=/usr/lib/coreos/etcd-wrapper $ETCD_OPTS \\\n --
name=\"controller\" \\\n --listen-peer-urls=\"http://
10.0.1.1:2380\" \\\n --listen-client-urls=\"http://
10.0.1.1:2379,http://127.0.0.1:2379\" \\\n --initial-advertise-
peer-urls=\"http://10.0.1.1:2380\" \\\n --advertise-client-
urls=\"http://10.0.1.1:2379\" \\\n —discovery=\"https://
discovery.etcd.io/...\""}]}]},"networkd":{"units":
[{"name":"static.network","contents":"[Match]
\nName=enp2s0\n\n[Network]
\nAddress=10.0.1.1/21\nGateway=10.0.0.1\nDNS=10.0.0.1\
n"}]},"passwd":{"users":[{"name":"core","sshAuthorizedKeys":
["ecdsa-sha2-nistp256 ..."]}]}}

Step 2:
Installing Kubernetes



There’s an official guide to deploy Kubernetes on Container Linux, and even though it works it’s a little out of date. But most importantly it’s laborious, and I’m sure there were scripts to automate it.



So I searched and found these. I thought they weren’t referenced anywhere in the Container Linux documentation, but I found them later

Kubernetes on CoreOS

Achieve high-availability and improve efficiency of container deployments

Kubernetes on CoreOS with Generic Install Scripts

This guide will setup Kubernetes on CoreOS in a similar way to other tools in the repo. The main goal of these scripts is to be generic and work on many different cloud providers or platforms. The notable difference is that these scripts are intended to be platform agnostic and thus don't automatically setup the TLS assets on each host beforehand.

While we provide these scripts and test them through the multi-node Vagrant setup, we recommend using a platform specific install method if available. If you are installing to bare-metal, you might find our baremetal repo more appropriate.

Generate TLS Assets

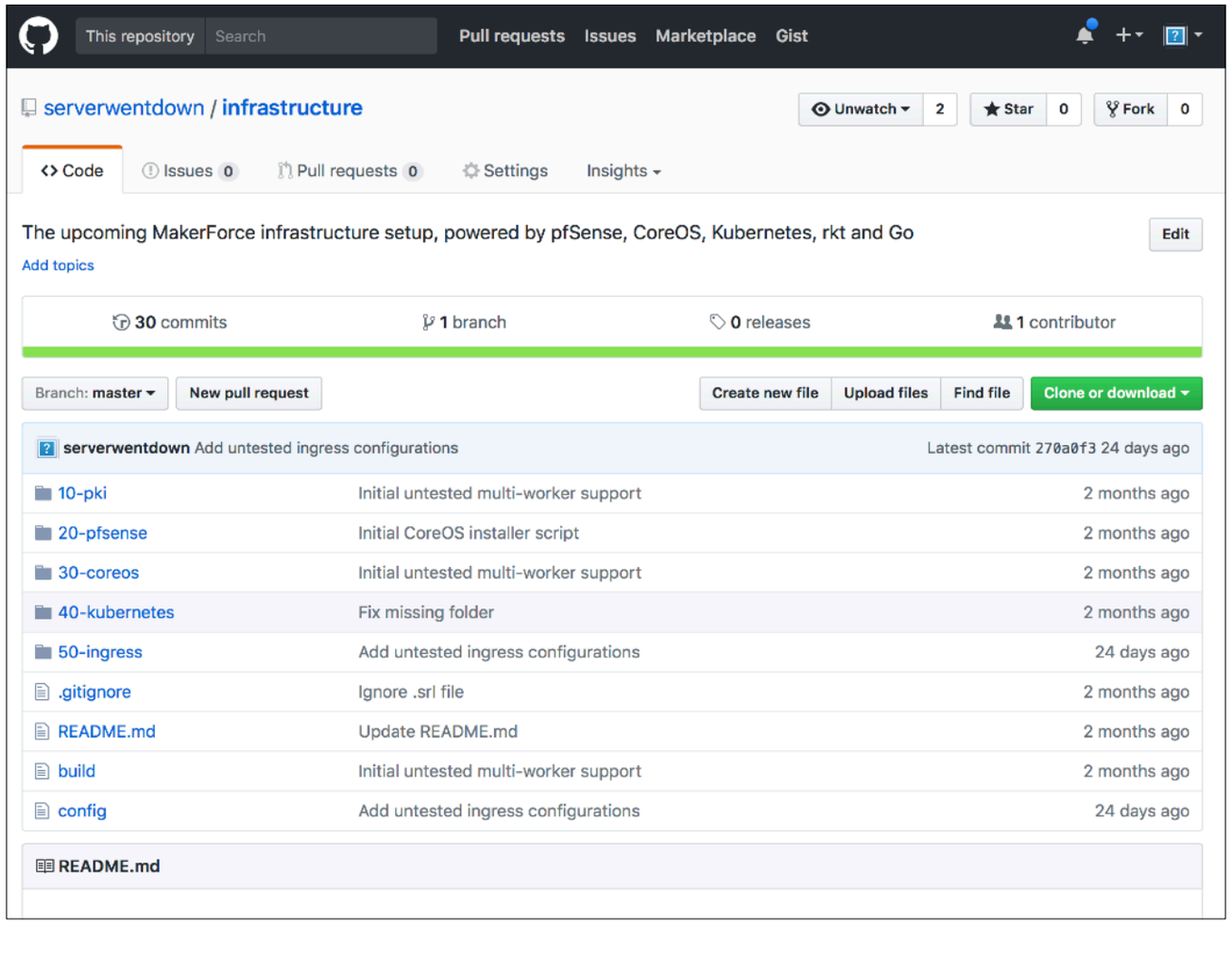
Review the OpenSSL-based TLS instructions for generating your TLS assets for each of the Kubernetes nodes.

Place the files in the following locations:

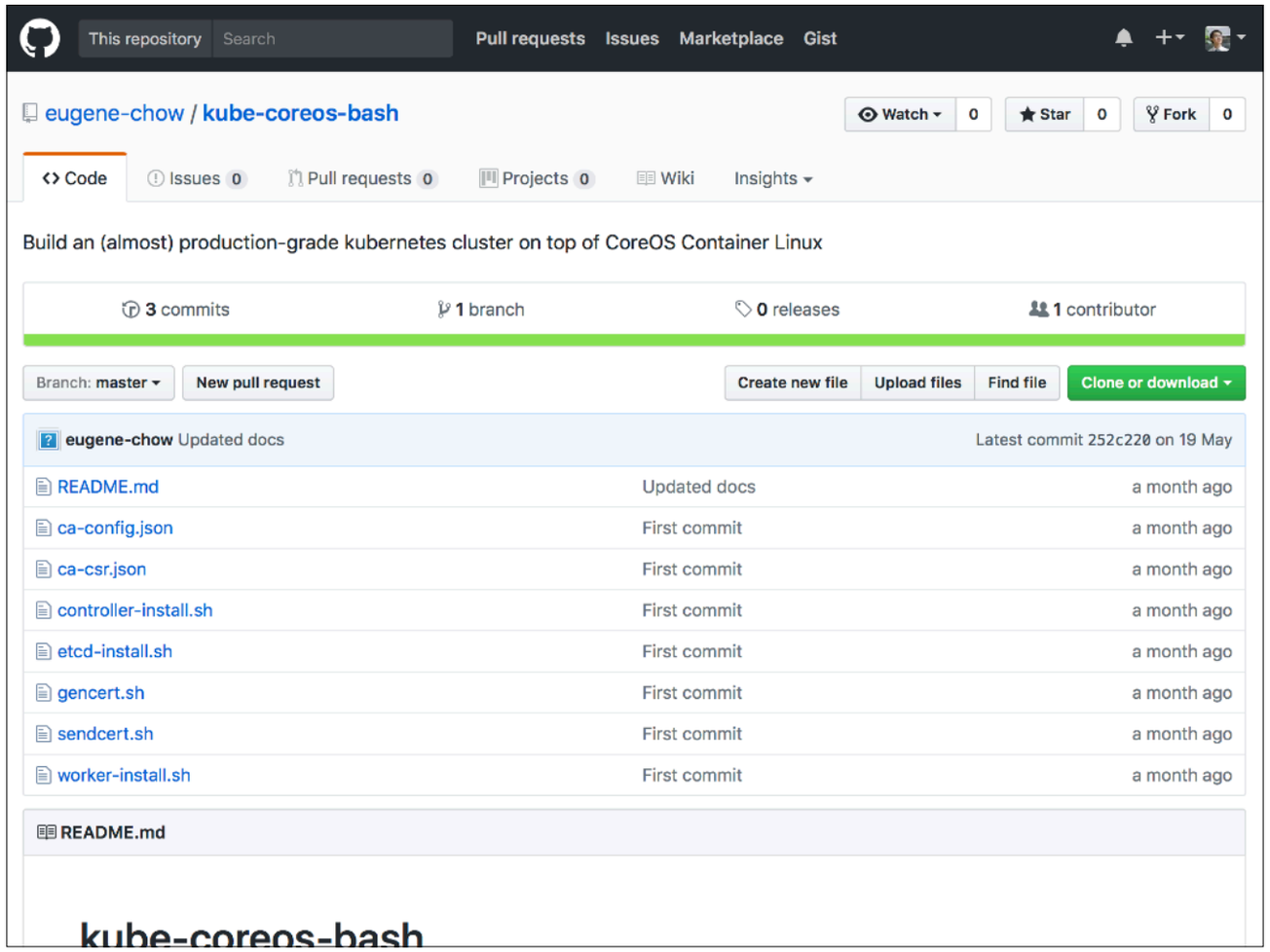
CONTROLLER FILES	LOCATION
API Certificate	`/etc/kubernetes/ssl/apiserver.pem`

Automating

So I knew that I could get Kubernetes up and running on baremetal, but I needed to be able to reproduce the entire setup without going through the entire process of editing scripts. Thus, I threw all the configuration into a single script that generates scripts to be run at each step of the process, and centralised the configuration, and of course put it on GitHub.



So here is my repo containing everything. It's still work in progress so don't look at it. Not even ready for a staging server. It's state is still experimental, but the eventual goal is to have an entire setup, including continuous integration, reverse proxying and management scripts in this repository, and make sure I can bring up an exact replica of my setup without effort.



There are also some projects out there like this one by Eugene Chow.

Why scripts?

There are other ways to install Kubernetes like through Tectonic or matchbox, but it requires me to modify my network setup, and requires at least three nodes, is not open source, so I didn’t manage to try it out.

Thank you!