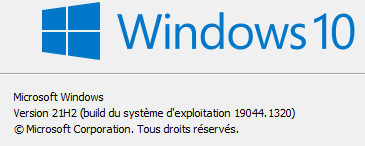
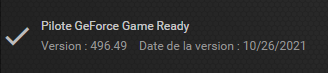
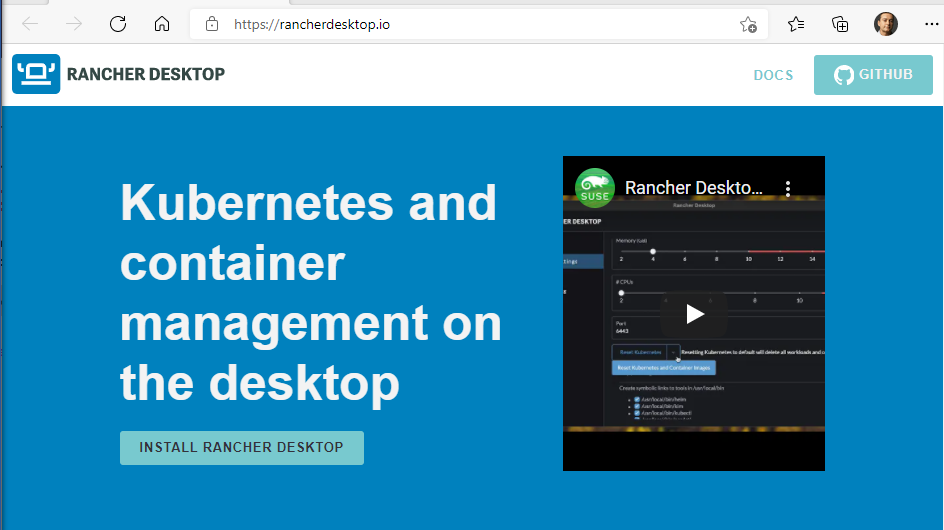
winver





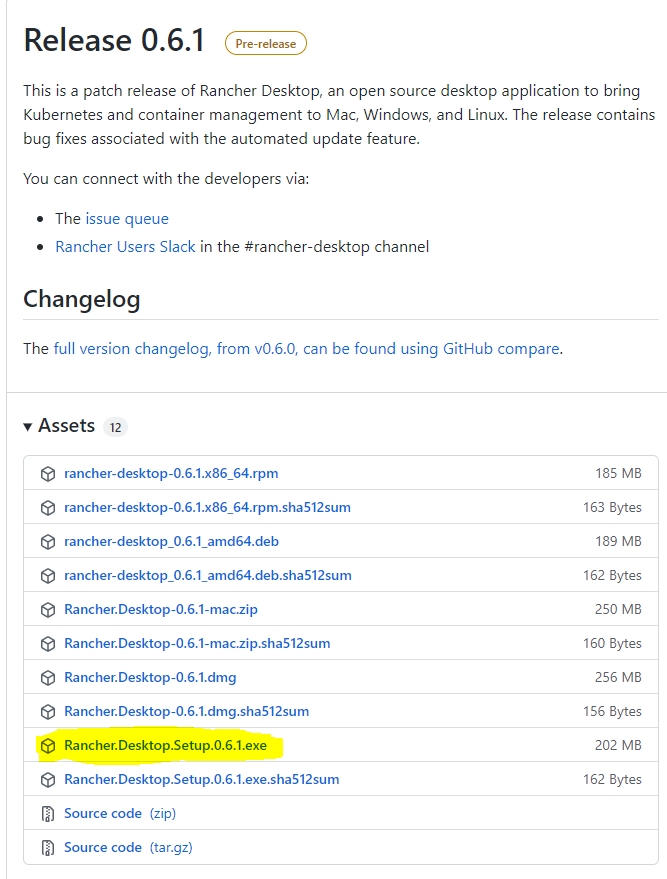
<https://rancherdesktop.io/>

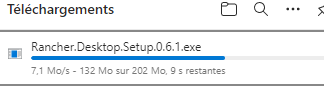


Click « Install Rancher Desktop »

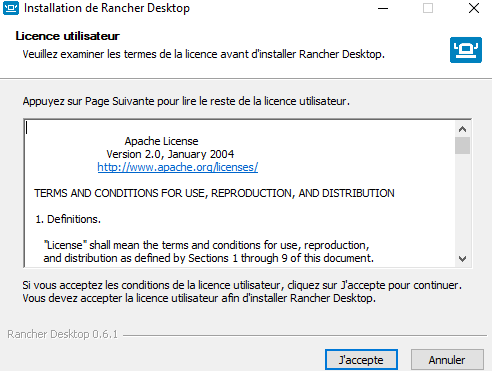
<https://github.com/rancher-sandbox/rancher-desktop/releases>

Expand « > Assets » in the latest release, then download and execute the .exe file :

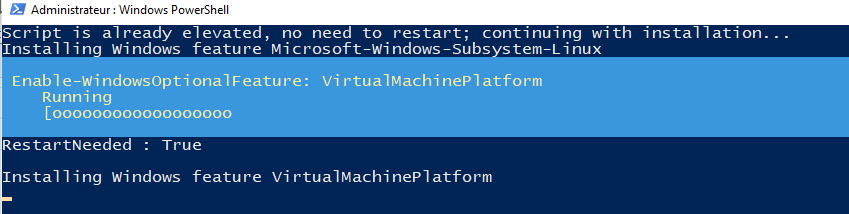




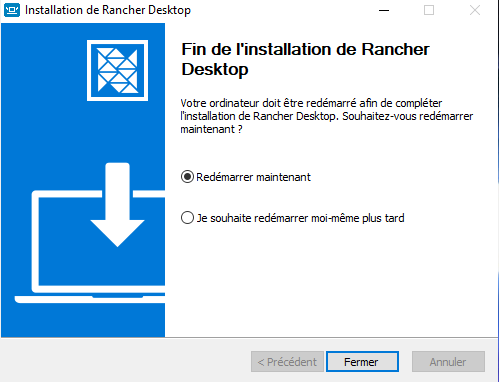
Accept the Apache licence :



Validate the elevated prompt to allow Powershell to install Windows Subsystem for Linux :



Restart your computer :

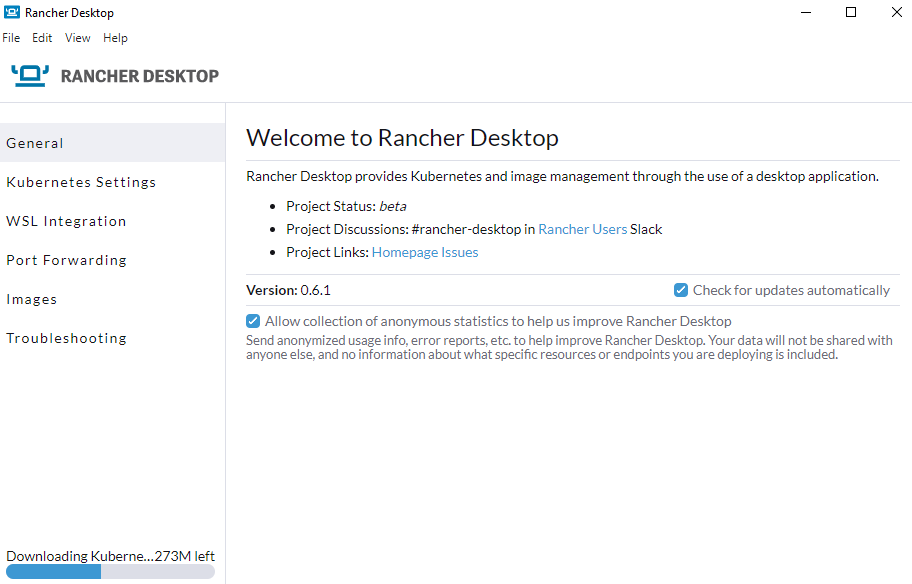


Launch Rancher Desktop for the first time :



Your computer may restart again to apply an update to Windows Subsystem for Linux.

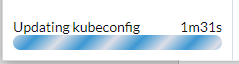
Launch Rancher Dekstop again :

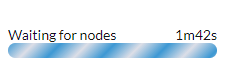












CPU and memory consumption :





Close with upper right cross :

Rancher Dekstop still running in the system tray :

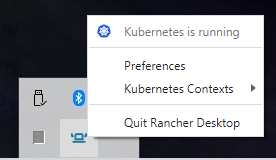


Memory consumption :





Right click on the icon to get commands menu :



Click on Preferences to reopen the UI.

Click on Quit Rancher Desktop to free resources :

* Rancher disappears instantaneously
* Vmmem disappears after 1 min inactivity

Double click on Rancher Desktop to relaunch.

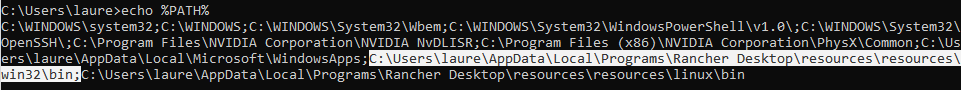
* Starting time : 18 sec



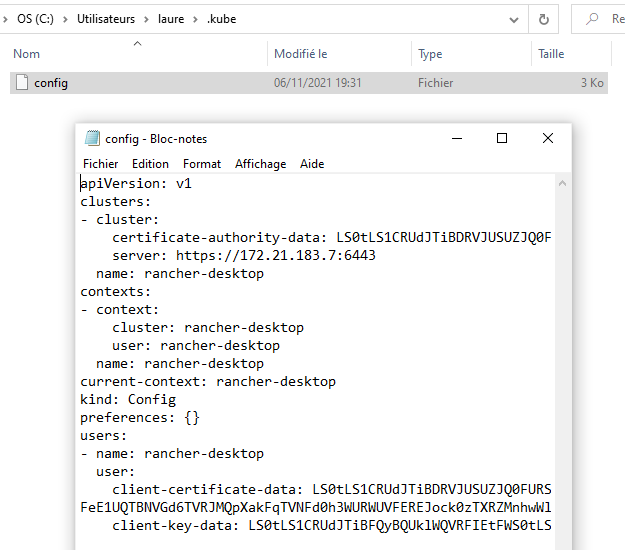
Disk space consumption :

1. C:\Users\laure\AppData\Local\Programs\Rancher Desktop : 751 Mo  
   - Rancher Desktop.exe : 125 Mo  
   - resources\resources\win32\bin : 168 Mo (kim/helm/kubectl/kubelr/nerdctl)  
   - resources\resources\linux\bin : 208 Mo (same)

NB : all these clients are added to the Windows PATH.

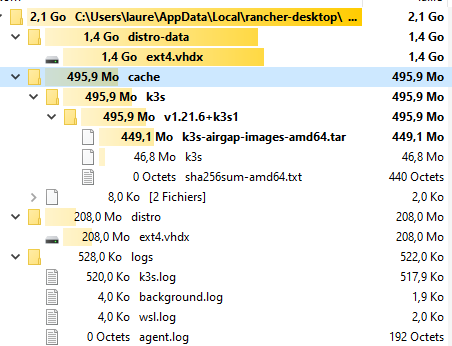


And a kubeconfig file was created in %USERPROFILE%\.kube\config :



Note : no .wslconfig was created in %USERPROFILE%.

1. C:\Users\laure\AppData\Local\rancher-desktop : 2.1 Go

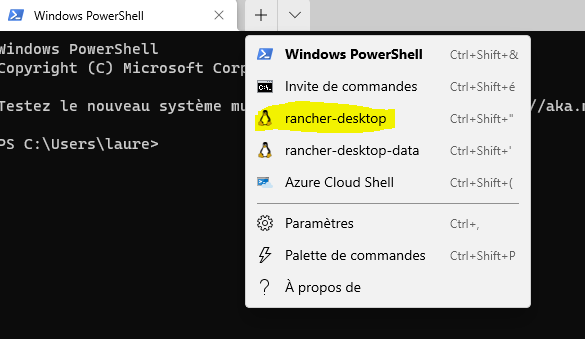


Access the distribution with Windows Terminal :

* Open the Microsoft Store and install Windows Terminal

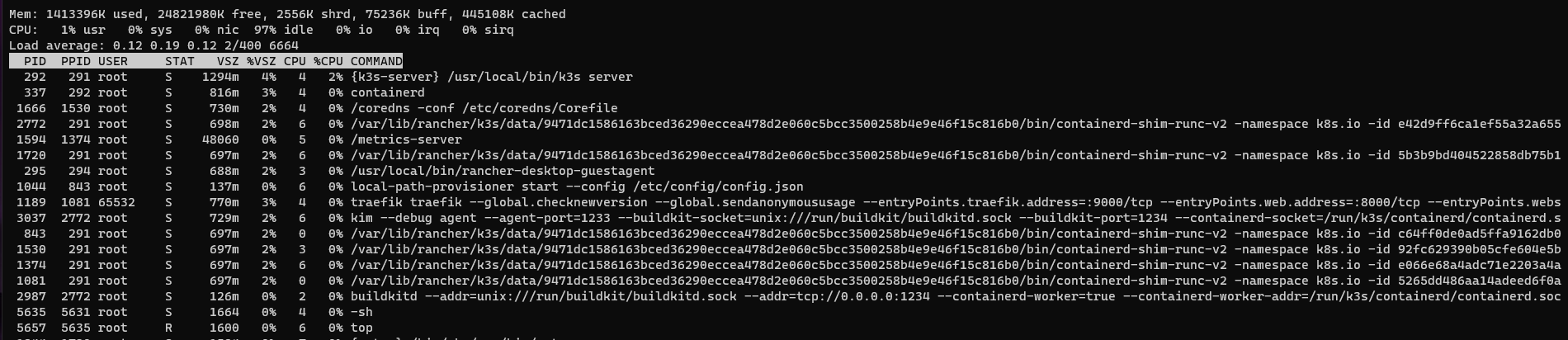


* Launch Windows Terminal, open rancher-desktop distribution :



top : 1.4 Go

k3s, containerd, coredns, traefik, local-path-provisioner, kim, buildkit, metrics-server, rancher-desktop-guest-agent



cat /etc/os-release

NAME="Rancher Desktop WSL Distro"

ID="rancher-desktop-wsl-distro"

PRETTY\_NAME="Rancher Desktop WSL Distribution"

VERSION\_ID="0.5"

BUILD\_ID="4d54b699fa59fc9580ef7023812807e79a76517e"

HOME\_URL="https://rancherdesktop.io/"

SUPPORT\_URL="https://rancher-users.slack.com/channels/rancher-desktop"

BUG\_REPORT\_URL="https://github.com/rancher-sandbox/rancher-desktop-wsl-distro/issues/new"

Distro : <https://github.com/rancher-sandbox/rancher-desktop-wsl-distro>

FROM alpine as builder

ADD build.sh /

ADD os-release /

ADD wsl.conf /

COPY nerdctl-${NERDCTL\_VERSION}.tgz /nerdctl.tgz

COPY rancher-desktop-guestagent-${AGENT\_VERSION} /rancher-desktop-guestagent

RUN /bin/sh /build.sh

FROM scratch

COPY --from=builder /distro/ /

<https://github.com/rancher-sandbox/rancher-desktop-wsl-distro/blob/main/build.sh>

# Bootstrap an alpine chroot in /distro

# Remove unnecessary packages

rm -f /distro/etc/apk/world

apk --root /distro --update-cache add --initdb busybox

apk --root /distro add ca-certificates

# Install nerdctl

tar -xvf /nerdctl.tgz -C /distro/usr/local/ \

bin/buildctl \

bin/buildkitd \

bin/nerdctl \

libexec/cni/bridge \

libexec/cni/portmap \

libexec/cni/firewall \

libexec/cni/tuning \

libexec/cni/isolation \

libexec/cni/host-local

# Add packages required for nerdctl

apk --root /distro add iptables ip6tables

# Add guest agent

chmod +x rancher-desktop-guestagent

cp rancher-desktop-guestagent /distro/usr/local/bin/

# Add Moby components

apk --root /distro add docker-engine

# Create the root user (and delete all other users)

echo root:x:0:0:root:/root:/bin/sh > /distro/etc/passwd

# Clean up apk metadata and other unneeded files

rm -rf /distro/var/cache/apk

rm -rf /distro/etc/network

# Generate /etc/os-release; we do it this way to evaluate variables.

# Configuration for WSL compatibility

cp wsl.conf /distro/etc/wsl.conf

# WSL configuration options for the Rancher Desktop distribution.

[automount]

# Prevent processing /etc/fstab, since it doesn't exist.

mountFsTab = false

ldconfig = false

~/tmp # cat /etc/alpine-release

3.14.2

Distrib : rancher-desktop-data is mounted in rancher-desktop at

/mnt/wsl/rancher-desktop/run/data

/dev/sdc on /mnt/wsl/rancher-desktop/run/data type ext4 (rw,relatime,discard,errors=remount-ro,data=ordered)

+ 2 other mount points :

/dev/sdc on /etc/rancher type ext4 (rw,relatime,discard,errors=remount-ro,data=ordered)

/dev/sdc on /var/lib type ext4 (rw,relatime,discard,errors=remount-ro,data=ordered)

\* Delete state related to Kubernetes. This will ensure that images are not

\* deleted.

\* @param execAsRoot A function to run commands on the VM as root.

\*/

async deleteKubeState(execAsRoot: (...args: string[]) => Promise<void>) {

const directories = [

'/var/lib/kubelet', // https://github.com/kubernetes/kubernetes/pull/86689

// We need to keep /var/lib/rancher/k3s/agent/containerd for the images.

'/var/lib/rancher/k3s/data',

'/var/lib/rancher/k3s/server',

'/var/lib/rancher/k3s/storage',

'/etc/rancher/k3s',

'/run/k3s',

];

console.log(`Attempting to remove K3s state: ${ directories.sort().join(' ') }`);

await Promise.all(directories.map(d => execAsRoot('rm', '-rf', d)));

<https://github.com/rancher-sandbox/rancher-desktop/tree/main/src/assets/scripts>

install-k3s

IMAGES="/var/lib/rancher/k3s/agent/images"

mkdir -p "${IMAGES}"

ln -s -f "${K3S\_DIR}/k3s-airgap-images-amd64.tar" "${IMAGES}"

ln -s -f "${K3S\_DIR}/k3s" /usr/local/bin

${K3S\_DIR} = C:\Users\laure\AppData\Local\rancher-desktop\cache\k3s\v1.21.6+k3s1

wsl-launch-k3s

exec /usr/local/bin/k3s server "$@"

then :

/usr/local/bin/launch-k3s --https-listen-port #desiredPort

Note : kubectl not installed but mounted in Linux distro

~ # which kubectl

/mnt/c/Users/laure/AppData/Local/Programs/Rancher Desktop/resources/resources/linux/bin/kubectl

~ # which helm

/mnt/c/Users/laure/AppData/Local/Programs/Rancher Desktop/resources/resources/linux/bin/helm

~ # which nerdctl

/usr/local/bin/nerdctl

But config not available => Solution to use kubectl from Linux :

* ln -s /mnt/c/Users/laure/.kube ~/.kube

Volumes

Annotations: pv.kubernetes.io/provisioned-by: rancher.io/local-path

Type: HostPath (bare host directory volume)

Path: /var/lib/rancher/k3s/storage/pvc-a92e5175-0fde-4a34-acb4-eff2cbfa8879\_default\_local-path-pvc

Test Longhorn

1. Install apk

[Index of /alpine/v3.14/main/x86\_64/ (alpinelinux.org)](http://dl-cdn.alpinelinux.org/alpine/v3.14/main/x86_64/)

wget http://dl-cdn.alpinelinux.org/alpine/v3.14/main/x86\_64/apk-tools-static-2.12.7-r0.apk

tar -zxvf apk-tools-static-2.12.7-r0.apk

sbin/apk.static -X http://dl-cdn.alpinelinux.org/alpine/latest-stable/main -U --allow-untrusted --initdb add apk-tools-static

apk.static update

sbin/apk.static -X http://dl-cdn.alpinelinux.org/alpine/latest-stable/main -U --allow-untrusted add apk-tools

apk update

1. Install <https://longhorn.io/docs/1.1.1/deploy/install/#installation-requirements>

apk add bash

apk add curl

apk add findmnt

apk add lsblk

apk add open-iscsi

~~apk add openrc~~

~~rc-update add iscsid~~

~~rc-service iscsid start~~

1. Install longhorn

kubectl apply -f https://raw.githubusercontent.com/longhorn/longhorn/v1.2.0/deploy/longhorn.yaml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: longhorn-volv-pvc

spec:

accessModes:

- ReadWriteOnce

storageClassName: longhorn

resources:

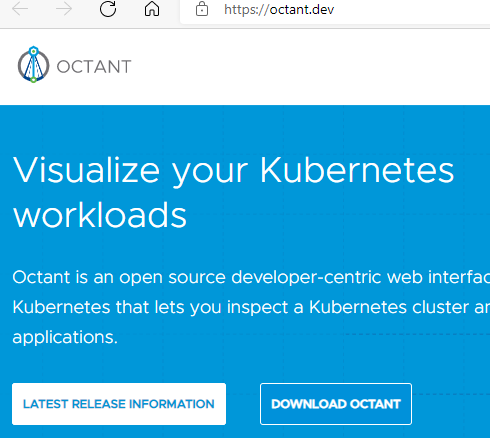
requests:

storage: 2Gi

Locate container images

/var/lib/rancher/k3s/agent/containerd

[Octant](https://octant.dev/)



<https://github.com/vmware-tanzu/octant/releases/v0.24.0>

Octant.Setup.0.24.0.exe

Just click on exe and everything works out of the box

**CREATE distro manually**

nerdctl run -it nvidia/cuda:11.2.0-base-ubuntu18.04 /bin/bash

# Set up environment

NVIDIA\_CONTAINER\_RUNTIME\_VERSION=3.5.0-1

echo 'debconf debconf/frontend select Noninteractive' | debconf-set-selections

mkdir -p /etc && echo 'hosts: files dns' > /etc/nsswitch.conf

chmod 1777 /tmp

PATH="$PATH:/bin/aux"

# Install NVIDIA Container Runtime

apt-get update && apt-get -y install gnupg2 curl wget

curl -s -L https://nvidia.github.io/nvidia-container-runtime/gpgkey | apt-key add -

curl -s -L https://nvidia.github.io/nvidia-container-runtime/ubuntu18.04/nvidia-container-runtime.list | tee /etc/apt/sources.list.d/nvidia-container-runtime.list

apt-get update && apt-get -y install nvidia-container-runtime=${NVIDIA\_CONTAINER\_RUNTIME\_VERSION}

# Provide custom containerd configuration to configure the nvidia-container-runtime

wget https://k3d.io/v5.0.3/usage/advanced/cuda/config.toml.tmpl

mkdir -p /var/lib/rancher/k3s/agent/etc/containerd/

mv config.toml.tmpl /var/lib/rancher/k3s/agent/etc/containerd/config.toml.tmpl

# Deploy the nvidia driver plugin on startup

wget https://k3d.io/v5.0.3/usage/advanced/cuda/device-plugin-daemonset.yaml

mkdir -p /var/lib/rancher/k3s/server/manifests

mv device-plugin-daemonset.yaml /var/lib/rancher/k3s/server/manifests/nvidia-device-plugin-daemonset.yaml

# Install nerdctl

NERDCTL\_VERSION=0.11.1

wget -O nerdctl.tgz "https://github.com/containerd/nerdctl/releases/download/v${NERDCTL\_VERSION}/nerdctl-full-${NERDCTL\_VERSION}-linux-amd64.tar.gz"

tar -xvf /nerdctl.tgz -C /usr/local/ \

bin/buildctl \

bin/buildkitd \

bin/nerdctl \

libexec/cni/bridge \

libexec/cni/portmap \

libexec/cni/firewall \

libexec/cni/tuning \

libexec/cni/isolation \

libexec/cni/host-local

# Add guest agent

AGENT\_VERSION=0.1.1

wget -O rancher-desktop-guestagent "https://github.com/rancher-sandbox/rancher-desktop-agent/releases/download/v${AGENT\_VERSION}/rancher-desktop-guestagent"

chmod +x rancher-desktop-guestagent

cp rancher-desktop-guestagent /usr/local/bin/

# Add Moby components

~~apk install docker-engine ???~~

# Create the root user (and delete all other users)

echo root:x:0:0:root:/root:/bin/sh > /etc/passwd

# Generate /etc/os-release; we do it this way to evaluate variables.

~~. /os-release~~

~~rm -f /etc/os-release # Remove the existing Ubuntu one~~

~~for field in $(awk -F= '/=/{ print $1 }' /os-release); do~~

~~value="$(eval "echo \${${field}}")"~~

~~if [ -n "${value}" ]; then~~

~~echo "${field}=\"${value}\"" >> /etc/os-release~~

~~fi~~

~~done~~

# Configuration for WSL compatibility

wget https://raw.githubusercontent.com/rancher-sandbox/rancher-desktop-wsl-distro/main/wsl.conf

cp wsl.conf /etc/wsl.conf

**START sequence**

WSLBackend

INSTANCE\_NAME = 'rancher-desktop';  
DATA\_INSTANCE\_NAME = 'rancher-desktop-data'

// The version of the WSL distro we expect  
DISTRO\_VERSION = '0.7';

upgradeDistroAsNeeded();

* if (isDistroRegistered(distribution = INSTANCE\_NAME))
  + wsl --list --quiet
  + distros.includes(distribution)
* existingVersion = getDistroVersion();
  + . /etc/os-release ; echo ${VERSION\_ID}
* if (semver.lt(existingVersion, desiredVersion, true))
* // Upgrading WSL distribution
* initDataDistribution()
  + see below
* execWSL('--unregister', INSTANCE\_NAME)

ensureDistroRegistered();

* if (!isDistroRegistered(distribution = INSTANCE\_NAME))
* // Registering WSL distribution
* mkdir(paths.wslDistro, { recursive: true });
* execWSL('--import', INSTANCE\_NAME, paths.wslDistro, this.distroFile, '--version', '2');
  + distroFile() => resources.get(os.platform(), `distro-${ DISTRO\_VERSION }.tar`)
    - path.join(process.resourcesPath, 'resources', ...pathParts)
  + Win32Paths. wslDistro()
    - APP\_NAME = 'rancher-desktop';
    - path.join(this.localAppData, APP\_NAME, 'distro');
* check isDistroRegistered()

initDataDistribution();

* workdir = mkdtemp(path.join(os.tmpdir(), 'rd-distro-'))
* archivePath = path.join(workdir, 'distro.tar')
* if (!isDistroRegistered({ distribution: DATA\_INSTANCE\_NAME }))
* // Initializing WSL data
* // Creating initial data distribution...
* // Create a distro archive from the main distro
* // Make sure all the extra data directories exist
* DISTRO\_DATA\_DIRS = [
* '/etc/rancher',
* '/var/lib',
* ];
* DISTRO\_DATA\_DIRS.map((dir) execCommand('/bin/busybox', 'mkdir', '-p', dir))
* // Figure out what required files actually exist in the distro; they
* // may not exist on various versions.
* REQUIRED\_FILES = [
* '/bin/busybox', // Base tools
* '/bin/mount', // Required for WSL startup
* '/bin/sh', // WSL requires a working shell to initialize
* '/lib', // Dependencies for busybox
* '/etc/wsl.conf', // WSL configuration for minimal startup
* '/etc/passwd', // So WSL can spawn programs as a user
* ];
* extraFiles = REQUIRED\_FILES.map(path => execCommand({ expectFailure: true }, 'busybox', '[', '-e', path, ']'); path;
* execCommand('tar', '-cf', wslify(archivePath), '-C', '/', ...extraFiles, ...DISTRO\_DATA\_DIRS);
* execWSL('--import', DATA\_INSTANCE\_NAME, paths.wslDistroData, archivePath, '--version', '2');

k3sHelper.ensureK3sImages(desiredVersion);

* appData = process.env['APPDATA'] || path.join(os.homedir(), 'AppData', 'Roaming')
* localAppData = process.env['LOCALAPPDATA'] || path.join(os.homedir(), 'AppData', 'Local')
* cache() = path.join(this.localAppData, APP\_NAME, 'cache');
* downloadUrl() = https://github.com/k3s-io/k3s/releases/download'
* cacheDir = path.join(paths.cache, 'k3s');
* filenames = {
* exe: 'k3s',
* images: 'k3s-airgap-images-amd64.tar',
* checksum: 'sha256sum-amd64.txt',
* }
* mkdir(cacheDir, { recursive: true })
* if (!verifyChecksums(path.join(cacheDir, fullVersion)))
* workDir =mkdtemp(path.join(cacheDir, `tmp-${ fullVersion }-`));
* fileURL = `${ this.downloadUrl }/${ fullVersion }/${ filename }`;
* outPath = path.join(workDir, filename);
* fetch(fileURL)
* verifyChecksums(workDir);
* safeRename(workDir, path.join(cacheDir, fullVersion))

mountData() ;

* mountRoot = '/mnt/wsl/rancher-desktop/run/data';
* // Only bind mount the root if it doesn't exist; because this is in the
* // shared mount (/mnt/wsl/), it can persist even if all of our distribution
* // instances terminate, as long as the WSL VM is still running.
* mountInfo = execWSL('--distribution', DATA\_INSTANCE\_NAME, '--exec', 'busybox', 'cat', '/proc/self/mountinfo');
* // Some times we can have the mount but the disk is missing.
* // In that case we need to umount it, and the re-mount.
* Try : execWSL('--distribution', DATA\_INSTANCE\_NAME, '--exec', 'busybox', 'test', '-e', device);
* Catch : execWSL('--distribution', DATA\_INSTANCE\_NAME, '--exec', 'busybox', 'umount', mountRoot);
* if (!hasValidMount)
* execWSL('--distribution', DATA\_INSTANCE\_NAME, 'mount', '--bind', '/', mountRoot);
* DISTRO\_DATA\_DIRS.map(dir => {
* execCommand('mkdir', '-p', dir);
* execCommand('mount', '-o', 'bind', `${ mountRoot }/${ dir.replace(/^\/+/, '') }`, dir);
* }

installTrivy() ;

* // Trivy (tri pronounced like trigger, vy pronounced like envy) is a simple and comprehensive scanner for vulnerabilities in container images, file systems, and Git repositories, as well as for configuration issues
* // download-resources.sh installed trivy into the resources area
* // This function moves it into /usr/local/bin/ so when trivy is
* // invoked to run through wsl, it runs faster.
* execCommand('mkdir', '-p', '/var/local/bin');
* wslInstall(trivyExecPath, '/usr/local/bin');

machineID = crypto.randomBytes(16).toString('hex');

* execCommand('/bin/sh', '-c', `echo '${ machineID }' > /tmp/machine-id`);
* execCommand('/bin/mv', '-n', '/tmp/machine-id', '/etc/machine-id');
* execCommand('/bin/rm', '-f', '/tmp/machine-id');

deleteIncompatibleData(desiredVersion);

* existingVersion = getPersistedVersion();
  + filepath = '/var/lib/rancher/k3s/version';
  + captureCommand({ expectFailure: true }, '/bin/cat', filepath))
* if (existingVersion)
* if (semver.gt(existingVersion, desiredVersion))
* // Deleting incompatible Kubernetes state due to downgrade from ${ existingVersion } to ${ desiredVersion }...
* k3sHelper.deleteKubeState()
  + directories = [
  + '/var/lib/kubelet', // https://github.com/kubernetes/kubernetes/pull/86689
  + // We need to keep /var/lib/rancher/k3s/agent/containerd for the images.
  + '/var/lib/rancher/k3s/data',
  + '/var/lib/rancher/k3s/server',
  + '/var/lib/rancher/k3s/storage',
  + '/etc/rancher/k3s',
  + '/run/k3s',
  + ];
  + directories.map(d => execAsRoot('rm', '-rf', d))

installCACerts() ;

* certs = https.globalAgent.options.ca;
* // <https://www.npmjs.com/package/win-ca>
* // Get Windows System Root certificates for Node.js.
* // On Windows, win-ca doesn't add CAs into the agent; rather, it patches
* // `tls.createSecureContext()` instead, so we don't have a list of CAs here.
* // We need to fetch it manually.
* rawCerts = WinCA({ generator: true, format: WinCA.der2.pem });
* certs.push(...rawCerts.map(cert => cert.replace(/\r/g, '')));
* workdir = mkdtemp(path.join(os.tmpdir(), 'rd-ca-'));
* execCommand('/bin/sh', '-c', 'rm -f /usr/local/share/ca-certificates/rd-\*.crt');
* certs.map(async(cert, index) => {
* filename = `rd-${ index }.crt`;
* stream.Readable.from(cert),
* fs.createWriteStream(path.join(workdir, filename), { mode: 0o600 }),
* execCommand('cp', wslify(path.join(workdir, filename)), '/usr/local/share/ca-certificates/');
* }
* rmdir(workdir, { recursive: true });
* execCommand('/usr/sbin/update-ca-certificates');

installK3s(desiredVersion);

* import INSTALL\_K3S\_SCRIPT from '@/assets/scripts/install-k3s';
* runInstallScript(INSTALL\_K3S\_SCRIPT, 'install-k3s', fullVersion, wslify(path.join(paths.cache, 'k3s'));
* VERSION="${1}"
* CACHE\_DIR="${2}"
* K3S\_DIR="${CACHE\_DIR}/${VERSION}"
* # Make sure any outdated kubeconfig file is gone
* mkdir -p /etc/rancher/k3s
* rm -f /etc/rancher/k3s/k3s.yaml
* # Add images
* IMAGES="/var/lib/rancher/k3s/agent/images"
* mkdir -p "${IMAGES}"
* ln -s -f "${K3S\_DIR}/k3s-airgap-images-amd64.tar" "${IMAGES}"
* # Add k3s binary
* ln -s -f "${K3S\_DIR}/k3s" /usr/local/bin
* chmod a+x "${K3S\_DIR}/k3s" || true

installWSLHelpers();

* import INSTALL\_WSL\_HELPERS\_SCRIPT from '@/assets/scripts/install-wsl-helpers';
* runInstallScript(INSTALL\_WSL\_HELPERS\_SCRIPT, 'install-wsl-helpers', wslify(resources.get('linux', 'bin', 'nerdctl-stub')));
* # The nerdctl shim must be setuid root to be able to create bind mounts within
* # /mnt/wsl so that nerdctl can see it.
* mkdir -p "/mnt/wsl/rancher-desktop/bin/"
* cp "${1}" "/mnt/wsl/rancher-desktop/bin/nerdctl"
* chmod u+s "/mnt/wsl/rancher-desktop/bin/nerdctl"

persistVersion(desiredVersion);

* filepath = '/var/lib/rancher/k3s/version';
* execCommand('/bin/sh', '-c', `echo '${ version }' > ${ filepath }`);

// Write the launch script.

installScriptWorkDir = mkdtemp(path.join(os.tmpdir(), 'rd-k3s-runner-'));

installScriptPath = path.join(installScriptWorkDir, 'launch-k3s');

writeFile(installScriptPath, LAUNCH\_K3S\_SCRIPT, 'utf-8');

execCommand('mv', wslify(installScriptPath), '/usr/local/bin/launch-k3s');

execCommand('chmod', 'a+x', '/usr/local/bin/launch-k3s');

rm(installScriptWorkDir, { recursive: true }

// Actually run K3s

args = ['--distribution', INSTANCE\_NAME, '--exec',

'/usr/bin/unshare', '--mount', '--propagation', 'private',

'/usr/local/bin/launch-k3s',

'--https-listen-port', this.#desiredPort.toString()];

options: childProcess.SpawnOptions = {

env: {

...process.env,

WSLENV: `${ process.env.WSLENV }:IPTABLES\_MODE:DISTRO\_DATA\_DIRS`,

DISTRO\_DATA\_DIRS: DISTRO\_DATA\_DIRS.join(':'),

IPTABLES\_MODE: 'legacy',

}

childProcess.spawn('wsl.exe', args, options);

* import LAUNCH\_K3S\_SCRIPT from '@/assets/scripts/wsl-launch-k3s';
* # This script is used to launch K3s on WSL2; this is necessary as we need to do
* # some mount namespace shenanigans, since we store the data on the WSL shared
* # mount (/mnt/wsl/rancher/desktop/) and that can have issues with lingering
* # tmpfs mounts after we exit. This means we need to run this script under
* # unshare (to get a private mount namespace), and then we can mark various
* # mount points as shared (for kim / buildkit). Kubelet will internally do some
* # tmpfs mounts for volumes (secrets, etc.), which will stay private and go away
* # once k3s exits, so that we can delete the data as necessary.
* for dir in / ${DISTRO\_DATA\_DIRS}; do
* mount --make-shared "${dir}"
* Done
* exec /usr/local/bin/k3s server "$@"

// Starting guest agent

launchAgent();

* # The Rancher Desktop agent runs in Windows Subsystem for Linux (WSL).
* # Currently, it detects ports that are routed via iptables and makes them available to be port forwarded by WSL.
* agentargs = ['--distribution', INSTANCE\_NAME, '--exec', '/usr/local/bin/rancher-desktop-guestagent'];
* this.agentprocess = childProcess.spawn('wsl.exe', agentargs);

// Waiting for Kubernetes API

k3sHelper.waitForServerReady(() => this.ipAddress, this.#desiredPort));

* socket = tls.connect({ host, port, rejectUnauthorized: false }
* cert = socket.getPeerCertificate();
* // Check that the certificate contains a SubjectAltName that
* // includes the host we're looking for; when the server starts, it
* // may be using an obsolete certificate from a previous run that
* // doesn't include the current IP address.
* … try again 1 sec later …
* console.log(`The K3s server is ready on ${ host }:${ port }.`);

// Updating kubeconfig

k3sHelper.updateKubeconfig(captureCommand(getWSLHelperPath(), 'k3s', 'kubeconfig')))

* contextName = 'rancher-desktop';
* … merge new context in kubeconfig …
  + candidatePaths = process.env.KUBECONFIG?.split(path.delimiter) || [];
  + for (const kubeConfigPath of candidatePaths) {
  + if (config.contexts.find(ctx => ctx.name === contextName)) {
  + return kubeConfigPath;
  + }
  + home = await this.findHome();
  + kubeDir = path.join(home, '.kube');
  + return path.join(kubeDir, 'config');
* console.log('Setting default context...');

// Waiting for services

client = new K8s.Client();

client.waitForServiceWatcher()

* // If this API call reports that there are zero services currently running,
* // return null (and it's up to the caller to retry later).
* // This doesn't make complete sense, because if we've reached this point,
* // the k3s server must be running. But with wsl we've observed that the service
* // watcher needs more time to start up. When this call returns at least one
* // service, it's ready.
* if ((await this.coreV1API.listServiceForAllNamespaces()).body.items.length === 0) {
* return null;
* }

// Trigger kuberlr to ensure there's a compatible version of kubectl in place

childProcess.spawnFile(resources.executable('kubectl'), ['config', 'current-context'],

// Waiting for nodes

client.waitForReadyNodes()

* \* Wait for at least one node in the cluster to become ready. This is taken
* \* as an indication that the cluster is ready to be used.
* nodes = this.coreV1API.listNode();

// --- on exit ---

stop();

* execWSL('--terminate', INSTANCE\_NAME);

// -- other ---

reset()

* execWSL('--unregister', DATA\_INSTANCE\_NAME);
* this.start(config);

factoryReset()

* this.del();
* [paths.cache, paths.config].map(dir => fs.promises.rm(dir, { recursive: true })))
* rmdir(paths.logs, { recursive: true });

del()

* this.stop();
* execWSL('--unregister', INSTANCE\_NAME);
* execWSL('--unregister', DATA\_INSTANCE\_NAME);

// --- NOT CALLED ! ---

async setIntegration(distro: string, state: boolean): Promise<string | undefined> {

* …
* const executable = await this.getWSLHelperPath();
* …
* const kubeconfigPath = await this.k3sHelper.findKubeConfigToUpdate('rancher-desktop');
* …
* console.log(`kubeconfig integration for ${ distro } set to ${ state }`);