Ubuntu on OCI Registries

Canonical Group Ltd

Dec 15, 2023
The Open Container Initiative (OCI) establishes standards for constructing container images that can be reliably installed across a variety of compliant host environments.

Ubuntu’s LTS Docker Image Portfolio provides OCI-compliant images that receive stable security updates and predictable software updates, thus ensuring consistency in both maintenance schedule and operational interfaces for the underlying software your software builds on.

Ubuntu OCI tarball is a minimal rootfs tarball ready for use to build OCI/Docker container base images. It is similar to Ubuntu Base but already contains the modifications needed to make the rootfs suitable for building OCI/Docker container images. It is available for the amd64, armhf, arm64, powerpc and ppc64el architectures. The rootfs tarballs are published under OCI partner images.

Canonical publishes official Docker images to Docker Hub based on OCI images that are built from the Ubuntu OCI rootfs tarballs. Images are also published to AWS ECR (Elastic Container Registry) gallery (in ubuntu and its namespaces), ACR (Azure Container Registry), OCIR (Oracle Container Infrastructure Registry), and there are plans to publish to more registries in the future.
BUILDING UBUNTU PRO OCI IMAGES

Similar to the Ubuntu Pro images in public clouds, one can build an Ubuntu Pro OCI image to leverage services like ESM (Extended Security Maintenance) and FIPS.

The easiest way to build an Ubuntu Pro container image is to make use of existing container management tools (like Docker) and enable the Pro services on top of an existing Ubuntu container image (e.g. ubuntu:focal).

Note: It is highly recommended that Ubuntu Pro container images should be built on hosts that are already covered by an Ubuntu Pro subscription.

This process is described in detail in the pro client documentation. The resulting Ubuntu Pro container image can then be loaded into your local Docker daemon (by using --load when running docker build) and can be deployed/published normally as any other container image.
Instructions for deploying Ubuntu Pro containers on Kubernetes and for creating a ‘chiselled’ Ubuntu base image are linked below:

• Deploy Ubuntu Pro containers on Kubernetes
• Create a chiselled Ubuntu base image for C/C++, Go and Rust applications
Ubuntu on OCI registries is a member of the Ubuntu family and the project warmly welcomes community projects, contributions, suggestions, fixes and constructive feedback.

- Code of conduct
- Get support
- Join our online chat
- Contribute to these docs

3.1 Deploy Ubuntu Pro containers on Kubernetes

You can easily deploy your Ubuntu Pro container images on Kubernetes clusters, provided that said clusters are composed of Ubuntu Pro nodes. Read more on the Ubuntu Pro website.

3.1.1 Prerequisites

You will need

- an Ubuntu Pro container image on a private registry. This can be built using the procedure given at Building Ubuntu Pro OCI images. Since the image is attached to your Ubuntu Pro subscription, it should be kept private to avoid the sharing of your Ubuntu Pro subscription with unwanted users.
- a Kubernetes cluster with an Ubuntu Pro subscription. To learn to deploy it for different clouds, refer to the options shown below:

Deploying Pro Kubernetes clusters in various clouds

EKS

Check out Deploy an Ubuntu Pro EKS cluster to learn how to deploy an Ubuntu Pro Kubernetes cluster on Elastic Kubernetes Service (EKS).

GCE

Check out Deploy Kubernetes with Ubuntu Pro on GCE to learn how to deploy an Ubuntu Pro Kubernetes cluster on Google Compute Engine (GCE).

OpenShift

See the OpenShift guide to learn how to create a cluster.

Limitations
Although OpenShift can be deployed on public clouds, it does not support having Ubuntu Pro instances as cluster nodes.

“RHCOS is the only supported operating system for OpenShift Container Platform control plane, or master, machines. While RHCOS is the default operating system for all cluster machines, you can create compute machines, which are also known as worker machines, that use RHEL as their operating system.”

—OpenShift docs

Get an Ubuntu Pro subscription

You can reach out to us to attach the cluster nodes to an Ubuntu Pro subscription, or contact rocks@canonical.com if you need additional support.

Success

After attaching your Ubuntu Pro subscription to the cluster nodes, you will have an Ubuntu Pro Kubernetes cluster running on OpenShift.

Tanzu

See the Tanzu guide to learn how to create a cluster.

Get an Ubuntu Pro subscription

If the resulting Kubernetes cluster is composed of Ubuntu nodes, you must attach them to a Pro subscription as described in this tutorial. Otherwise, if the Kubernetes cluster is not running on Ubuntu nodes, you can contact us to attach the nodes to an Ubuntu Pro subscription, or contact rocks@canonical.com if you need additional support.

Success

After attaching your Ubuntu Pro subscription to the cluster nodes, you will have an Ubuntu Pro Kubernetes cluster running on Tanzu.

Nutanix

See the Nutanix Karbon Guide to learn how to create a cluster.

Limitations

Nutanix does not have Ubuntu Pro offerings for the nodes, i.e. you cannot choose Ubuntu Pro images for the nodes.

“Deploying Kubernetes clusters in Karbon requires a CentOS image. You must choose from a CentOS version and download the image.”

—Nutanix docs: “Downloading Images”

Get an Ubuntu Pro subscription

You can contact us to attach the cluster nodes to an Ubuntu Pro subscription, or contact rocks@canonical.com if you need additional support.

Success

After attaching your Ubuntu Pro subscription to the cluster nodes, you will have an Ubuntu Pro Kubernetes cluster running on Nutanix.
3.1.2 Create a Secret for Private Registry

Since your Ubuntu Pro container image is in a private registry, you will need to create a secret in Kubernetes (For more details about pulling images from private registries, check out the Kubernetes documentation).

You can use the following example command to create a secret named `regcred` for Docker Hub (i.e. using `https://index.docker.io/v1/` as `<your-registry-server>`).

```
kubectl create secret docker-registry regcred \
   --docker-server=<your-registry-server> \
   --docker-username=<your-username> \
   --docker-password=<your-password> \
   --docker-email=<your-email>
```

3.1.3 Deploy Pro container image to Pro Kubernetes cluster

You can deploy your Pro container image in a Pod, Deployment, or as a Service. Make sure to include your created secret and your Pro container image correctly.

Here is a manifest for a Pod that consists of a container running your Ubuntu Pro image. It uses your secret `regcred` to pull the Pro container image from your private registry.

```
# pro-container-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pro-container-pod
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: ubuntu-pro-container
      image: <your-private-pro-image>
      restartPolicy: OnFailure
```

Replace `<your-private-pro-image>` with your private Pro container image (something similar to `janedoe/jdoe-private:v1` for Docker Hub).

Create the Pod, and verify that the Pod is running:

```
kubectl apply -f pro-container-pod.yaml
kubectl get pod pro-container-pod
```

3.1.4 Check pod logs

Your Pro container image is deployed in the Pro Kubernetes cluster and running inside a Pod. You can check the logs by running:

```
kubectl logs pod/pro-container-pod
```
3.2 Create a chiselled Ubuntu base image for C/C++, Go and Rust applications

This guide will provide step-by-step instructions on how to create your own chiselled Ubuntu container image to run a compiled application.

- Chiselled Ubuntu are appliance-type container images combining both the advantages of distroless and Ubuntu to create smaller, more secure containers, without loosing the value add of a stable Linux distribution.
- The reduced size of the containers reduces the overall attack surface. Combined with the support and content quality from the Ubuntu distribution, chiselled Ubuntu is a significant security improvement.
- Chisel provides a developer-friendly CLI to install slices of packages from the upstream Ubuntu distribution onto the container filesystem.

3.2.1 Build the chiselled Ubuntu base image

This image must contain all the essential slices that are required for the execution of common compiled applications. As a base, the must-have list of slices is:

- `base-files_base` and `base-files_release-info`: will give you the overall base structure of the container image’s filesystem, together with the underlying Ubuntu base’s release information,
- `ca-certificates_data`: for cryptographic certificate verifications,
- `libgcc-s1_libs` and `libc6_libs`: for the libgcc-s1 and libc6 libraries.

Additionally, if you need SSL for your applications, make sure to also install the `openssl_config` slice. Here are other slices that might be useful (or even needed) depending on your target application’s requirements:

- `tzdata_zoneinfo`: for the timezone database,
- `libc-bin_nsswitch`: used by some applications to obtain name-service information.

For this guide, let’s use Ubuntu Jammy as the reference Ubuntu release for the target chiselled image.

Start by creating a Dockerfile with the following content:

ARG UBUNTU_RELEASE=22.04
ARG ARCH

# Build the chiselled filesystem based on the desired slices.
FROM ubuntu:$UBUNTU_RELEASE AS builder
ARG UBUNTU_RELEASE
ARG ARCH

# Get chisel binary
ADD "https://github.com/canonical/chisel/releases/download/v0.8.0/chisel_v0.8.0_linux_...
ARCH.tar.gz" chisel.tar.gz

RUN tar -xvf chisel.tar.gz -C /usr/bin/

RUN apt-get update \
    && DEBIAN_FRONTEND=noninteractive apt-get install -y ca-certificates

COPY --from=chisel /opt/chisel/chisel /usr/bin/

WORKDIR /rootfs

RUN chisel cut --release ubuntu-$UBUNTU_RELEASE --root /rootfs \
    base-files_base \n    base-files_release-info \n
(continues on next page)
Build the chiselled filesystem the only thing present in the final image.

```bash
FROM scratch
COPY --from=builder /rootfs /
```

Build the chiselled Ubuntu base image by running:

```bash
docker build -t chiselled-ubuntu-base:latest . --build-arg ARCH=<your_arch> # example
    ...ARCH=amd64
```

You’ll then find yourself with a new container image with approximately 5MB (or 2.5MB when compressed).

### 3.2.2 Build the application image

Now that you have the chiselled-ubuntu-base:latest image, you can simply add your compiled application to the image and run it from there. For the sake of simplicity, this guide will give your three very simple “Hello World” application examples for C, Go and Rust.

**C**

Assume the following `app.c` source code:

```c
#include <stdio.h>

int main() {
    printf("Hello World!");
    return 0;
}
```

You’d typically compile this via `gcc app.c -o app`.

**Go**

Assume the following `app.go` source code:

```go
package main
import "fmt"

func main() {
    fmt.Println("Hello World!")
}
```

You’d typically compile this via `go build -o app app.go`.

**Rust**

Assume the following `app.rs` source code:

```rust
fn main() {
    println!("Hello World!");
}
```
Ubuntu on OCI Registries

You’d typically compile this via `rustc app.rs`.

To build the final application image, you simply need to add your compiled executable to the `chiselled-ubuntu-base:latest` container image. So your new Dockerfile should be similar to:

```
FROM chiselled-ubuntu-base:latest
COPY app /
ENTRYPOINT ["./app"]
```

Build this chiselled application image with `docker build -t chiselled-app:latest .` and then run it:

```
docker run chiselled-app:latest
```

And the output should be:

```
Hello World!
```

### 3.2.3 What’s achieved?

The demonstrated chiselled Ubuntu image provides a sub-5MB runtime container image for your C/C++, Go and Rust applications whilst still allowing for additional slices to be easily added to cope with more complex use cases.

Chiselled Ubuntu images offer the benefits of a well-known and well-maintained Linux distribution combined with the advantages of ultra-small distroless-type container images, offering a secure and efficient foundation for building and deploying containerised applications.

### 3.3 Contribute to these docs

These docs are on located on a GitHub repository at: [ubuntu-cloud-docs](https://github.com/ubuntu-cloud-docs) and you’ll need a GitHub account to make contributions. It is a good idea to fork this repository into your own account before you start, otherwise GitHub will anyway prompt you to do so when you attempt your first change.

The docs are:

- structured using the Diátaxis approach,
- written in `reStructuredText` as per the Canonical style guide,
- built with Sphinx and
- hosted on Read the Docs.

We are always looking for ways to improve our docs, so we appreciate your contributions!

#### 3.3.1 Minor changes

If you’ve found a problem that can be fixed with a small change, you can use the pencil icon at the top of the relevant page to edit it on GitHub. When you are done with your edits, select Commit changes… on the top right. This will help you create a new branch and start a pull request (PR). Use Propose changes to submit the PR. We will review it and merge the changes.
3.3.2 Suggestions and questions

Use the Give feedback button at the top of any page to create a GitHub issue for any suggestions or questions that you might have.

3.3.3 New content

While contributing new content, it is easier to work with the docs on your local machine. You can submit a PR after all the checks have passed and things looks satisfactory.

Download and install the docs

If you are working with these docs for the first time, you’ll need to create a fork of the ubuntu-cloud-docs repository on your GitHub account and then clone that fork to your local machine. Once cloned, go into the ubuntu-cloud-docs directory and run:

```
make install
```

This creates a virtual environment and installs all the dependencies specified in .sphinx/requirements.txt. You only have to do this step once, and can skip it the next time you want to contribute.

Build and serve the docs

Use the make run command to build and serve the docs at 127.0.0.1:8000 or equivalently at localhost:8000. This gives you a live preview of the changes that you make (and save), without the need for a rebuild:

```
PROJECT=oci make run
```

Setting the PROJECT parameter to oci ensures that the documentation set for Ubuntu on OCI Registries gets built. This parameter is needed to distinguish between the different documentation sets present in the repository.

Perform checks and submit PR

Before opening a PR, run the following checks and also ensure that the documentation builds without any warnings (warnings are treated as errors in the publishing process):

```
PROJECT=oci make spelling
PROJECT=oci make linkcheck
PROJECT=oci make woke
```

If you need to add new words to the allowed list of words, include them in .wordlist.txt.

Once all the edits are done, commit the changes and push it to your fork. From the GitHub GUI of your fork, select the commit and open a PR for it.