Table of Contents

$ buildah

Advanced

Build OCI container images without a daemon

The buildah command builds OCI and Docker container images without requiring a running daemon. Part of the Podman container ecosystem developed by Red Hat, Buildah provides fine-grained control over the image build process - either using standard Dockerfiles or through scripted builds that manipu...

Options & Flags

`from`	Create a working container from a base image
`run`	Run a command inside the working container
`copy`	Copy files into the working container
`config`	Set image configuration (CMD, ENV, EXPOSE, etc.)
`commit`	Create an image from a working container
`bud (build-using-dockerfile)`	Build image from Dockerfile
`mount`	Mount working container filesystem on host
`unmount`	Unmount a working container filesystem
`images`	List locally stored images
`rm`	Remove a working container

Practical Examples

Example: Scripted build (no Dockerfile)

```
$ ctr=$(buildah from alpine:3.19) && buildah run $ctr apk add --no-cache python3 pip && buildah copy $ctr ./app /app && buildah co
```

Build an image step-by-step using shell commands. Each command gives you full control over the layer.

Example: Build from Dockerfile

```
$ buildah bud -t myapp:v1.0 --layers --target production .
```

Standard Dockerfile build with layer caching and multi-stage target. Compatible with docker build.

Example: Build from scratch (empty image)

```
$ ctr=$(buildah from scratch) && mnt=$(buildah mount $ctr) && cp mybinary $mnt/ && buildah config --cmd "/mybinary" $ctr && buildal
```

Create a minimal image from scratch containing only your static binary. Smallest possible image size.

Example: Mount and inspect filesystem

```
$ ctr=$(buildah from nginx:alpine) && mnt=$(buildah mount $ctr) && ls $mnt/etc/nginx/ && buildah unmount $ctr && buildah rm $ctr
```

Mount a container filesystem on the host for inspection, file copying, or debugging.

Example: Push to registry

```
$ buildah push myapp:latest docker://docker.io/myuser/myapp:latest
```

Push a locally built image to Docker Hub. Supports any OCI-compatible registry.

Tips & Best Practices

Pro Tip: Rootless builds in CI/CD: Buildah builds images without a daemon or root privileges - perfect for CI/CD pipelines (GitHub Actions, GitLab CI, Jenkins) where running dockerd is impractical.

Note: podman build uses Buildah: podman build is a wrapper around Buildah for Dockerfile builds. Use buildah directly when you need scripted builds, filesystem mounting, or building from scratch.

Warning: Working containers are temporary: Working containers created with buildah from are temporary. Always commit (buildah commit) before removing them, or your work is lost.

Pro Tip: Use --layers for caching: Use buildah bud --layers to enable layer caching during Dockerfile builds. This dramatically speeds up rebuilds when only later layers change.

$ docker

Intermediate

Container runtime and management tool

The docker command is the primary CLI for Docker - the industry-standard platform for building, shipping, and running containerized applications. Docker enables you to package applications with all their dependencies into lightweight, portable containers that run consistently across any environme...

Options & Flags

`run`	Create and start a container from an image
`ps`	List running containers (-a for all)
`build`	Build an image from a Dockerfile
`pull`	Download an image from a registry
`push`	Upload an image to a registry
`exec`	Execute a command inside a running container
`logs`	View container stdout/stderr logs
`stop/start/restart`	Manage container lifecycle
`compose up -d`	Start multi-container applications from compose.yaml
`system prune`	Remove unused containers, images, networks, and volumes

Practical Examples

Example: Run a container

```
$ docker run -d --name webserver -p 8080:80 -v ./html:/usr/share/nginx/html nginx:alpine
```

Run an Nginx container: detached (-d), named, port-mapped, with a bind mount for serving local HTML files.

Example: Build an image from Dockerfile

```
$ docker build -t myapp:v1.0 --target production .
```

Build a Docker image from the current directory Dockerfile, targeting the production stage in multi-stage build.

Example: Execute shell in running container

```
$ docker exec -it webserver sh
```

Open an interactive shell inside a running container for debugging.

Example: View and follow logs

```
$ docker logs -f --since 5m webserver
```

Follow container logs from the last 5 minutes. Useful for debugging application issues.

Example: Copy files between host and container

```
$ docker cp webserver:/etc/nginx/nginx.conf ./nginx.conf
```

Copy the Nginx config file from inside the container to the host filesystem.

Tips & Best Practices

Pro Tip: Always pin image versions: Never use :latest in production. Pin versions like nginx:1.25-alpine or postgres:16.3-alpine to ensure reproducible builds.

Warning: docker system prune is destructive: docker system prune -a --volumes removes ALL unused images, containers, and volumes. This includes stopped containers and their data. Always check with docker ps -a first.

Note: Use --rm for temporary containers: For one-off commands or testing, use docker run --rm to automatically remove the container when it exits. Prevents accumulation of stopped containers.

Pro Tip: Multi-stage builds: Use multi-stage Dockerfiles to keep production images small. Build in one stage, copy only the binary/output to a minimal runtime stage (alpine or distroless).

$ helm

Intermediate

Kubernetes package manager

The helm command is the package manager for Kubernetes, equivalent to apt for Debian or dnf for Fedora. Helm uses a packaging format called Charts - collections of templated Kubernetes manifests that can be versioned, shared, and deployed with customizable values.

Helm simplifies deploying compl...

Options & Flags

`install NAME CHART`	Install a chart as a named release
`upgrade NAME CHART`	Upgrade an existing release
`uninstall NAME`	Remove a release from the cluster
`list`	List all releases in the current namespace
`rollback NAME REVISION`	Rollback a release to a previous revision
`repo add/update`	Add or update chart repositories
`search repo/hub`	Search for charts in repos or Artifact Hub
`template`	Render chart templates locally without installing
`package`	Package a chart directory into a .tgz archive
`lint`	Check a chart for issues

Practical Examples

Example: Install PostgreSQL from Bitnami

```
$ helm repo add bitnami https://charts.bitnami.com/bitnami && helm install mydb bitnami/postgresql --set auth.postgresPassword=sec
```

Add the Bitnami repo and install PostgreSQL with a custom password in the database namespace.

Example: Install with custom values file

```
$ helm install myapp ./mychart -f values-production.yaml -n production
```

Install a local chart with environment-specific values. Values file overrides defaults in values.yaml.

Example: Upgrade a release

```
$ helm upgrade myapp ./mychart --set image.tag=v2.0 --set replicaCount=3
```

Upgrade with new image tag and replica count. Creates a new revision for rollback.

Example: Rollback after failed upgrade

```
$ helm history myapp && helm rollback myapp 2
```

View release history and rollback to revision 2. Kubernetes recreates the previous state.

Example: Preview templates before install

```
$ helm template myapp ./mychart -f values-prod.yaml --debug
```

Render all Kubernetes manifests locally without installing. Essential for validating templates.

Tips & Best Practices

Pro Tip: Use --dry-run for safety: Before installing or upgrading, use helm install --dry-run to simulate the operation and see what would be applied without making changes.

Warning: Pin chart versions: Always specify --version when installing charts in production: helm install mydb bitnami/postgresql --version 15.2.5. Without it, you get the latest version which may include breaking changes.

Note: upgrade --install is idempotent: Use helm upgrade --install to either install (if not present) or upgrade (if present). This is the recommended pattern for CI/CD pipelines.

Pro Tip: Use helm diff plugin: Install the helm-diff plugin to see exactly what would change before upgrading: helm diff upgrade myapp ./mychart -f values.yaml

$ kubectl

Intermediate

Kubernetes cluster management CLI

The kubectl command is the official command-line tool for interacting with Kubernetes clusters. It communicates with the Kubernetes API server to deploy applications, inspect and manage cluster resources, view logs, and execute commands inside containers running on the cluster.

kubectl is the Sw...

Options & Flags

`get`	List resources (pods, services, deployments, etc.)
`describe`	Show detailed information about a resource
`apply -f`	Apply a configuration file (declarative)
`delete`	Delete resources by name or from file
`logs`	View container logs (supports -f for follow)
`exec`	Execute a command in a container
`scale`	Scale a deployment to N replicas
`rollout`	Manage deployment rollouts (status, history, undo)
`port-forward`	Forward a local port to a pod
`config`	Manage kubeconfig (contexts, clusters, users)

Practical Examples

Example: List pods in all namespaces

```
$ kubectl get pods --all-namespaces -o wide
```

Show all pods across all namespaces with node assignment and IP addresses.

Example: Deploy from YAML manifest

```
$ kubectl apply -f deployment.yaml -f service.yaml -f ingress.yaml
```

Declaratively apply multiple Kubernetes manifests. Creates or updates resources to match the desired state.

Example: Debug a failing pod

```
$ kubectl describe pod myapp-abc123 -n production && kubectl logs myapp-abc123 -n production --previous
```

Show pod events and conditions, then view logs from the previous (crashed) container instance.

Example: Shell into a running pod

```
$ kubectl exec -it myapp-abc123 -- /bin/sh
```

Open an interactive shell inside a running container for debugging. Use -- to separate kubectl args from the command.

Example: Scale a deployment

```
$ kubectl scale deployment myapp --replicas=5 && kubectl rollout status deployment/myapp
```

Scale to 5 replicas and wait for the rollout to complete.

Tips & Best Practices

Pro Tip: Use aliases: Set alias k=kubectl and enable kubectl completion in bash/zsh for dramatically faster command entry. Add to .bashrc: source <(kubectl completion bash)

Warning: Check context before destructive ops: Always verify your current context before delete/apply: kubectl config current-context. Accidentally deleting resources in production is a real risk.

Note: Declarative > Imperative: Use kubectl apply -f manifest.yaml (declarative) in production, not kubectl run/create (imperative). Store manifests in git for versioning and reproducibility.

Pro Tip: Use --dry-run for testing: Generate YAML without applying: kubectl create deployment myapp --image=nginx --dry-run=client -o yaml > deployment.yaml

$ podman

Intermediate

Daemonless rootless container management tool

The podman command is a next-generation container engine that manages OCI containers without requiring a daemon or root privileges. Developed by Red Hat, Podman provides a Docker-compatible CLI while offering superior security through its rootless-first, daemonless architecture.

Unlike Docker, w...

Options & Flags

`run`	Create and start a container (rootless by default)
`pod create`	Create a new pod (shared network namespace)
`generate kube`	Generate Kubernetes YAML from a pod/container
`play kube`	Deploy containers from Kubernetes YAML
`generate systemd`	Generate systemd unit files for a container
`build`	Build an image using Buildah backend
`ps`	List containers (-a for all, --pod for pod info)
`exec`	Execute command inside running container
`auto-update`	Automatically update container images
`machine init/start`	Initialize and start Podman VM on macOS/Windows

Practical Examples

Example: Run rootless container

```
$ podman run -d --name myapp -p 8080:80 nginx:alpine
```

Run a container as your regular user without sudo. Rootless by default - no daemon, no root privileges.

Example: Create and use a pod

```
$ podman pod create --name webapp -p 8080:80 -p 5432:5432 && podman run -d --pod webapp --name web nginx:alpine && podman run -d -
```

Create a pod and add web + database containers sharing the same network namespace (localhost communication).

Example: Generate Kubernetes YAML

```
$ podman generate kube webapp > webapp.yaml
```

Export a running pod as Kubernetes-compatible YAML. Deploy to K8s with kubectl apply -f webapp.yaml.

Example: Run container as systemd service

```
$ podman generate systemd --name web --files --new && cp container-web.service ~/.config/systemd/user/ && systemctl --user enable
```

Generate a systemd unit file and install it as a user service. Container auto-starts on boot.

Example: Use Docker Compose with Podman

```
$ systemctl --user enable --now podman.socket && export DOCKER_HOST=unix:///run/user/$(id -u)/podman/podman.sock && docker compose
```

Enable the Podman socket for Docker API compatibility. Docker Compose works transparently.

Tips & Best Practices

Pro Tip: alias docker=podman: Podman is CLI-compatible with Docker. Add alias docker=podman to your .bashrc for seamless migration. Most Docker commands work identically.

Note: Rootless ports below 1024: Rootless containers cannot bind to ports below 1024 by default. Fix with: sysctl net.ipv4.ip_unprivileged_port_start=80 or use port mapping like -p 8080:80.

Warning: Storage location differs: Rootless Podman stores images in ~/.local/share/containers/ (not /var/lib/docker). Root and rootless have completely separate image stores.

Pro Tip: Use Quadlet for production services: Instead of generate systemd, use Quadlet .container files in /etc/containers/systemd/ for a cleaner, more maintainable systemd integration.

$ skopeo

Intermediate

Inspect and copy container images without a daemon

The skopeo command inspects, copies, and manages container images across registries without requiring a running container daemon or pulling the images locally. Part of the Podman ecosystem developed by Red Hat, Skopeo operates directly on container registries, making it ideal for CI/CD pipelines,...

Options & Flags

`inspect`	Inspect image metadata without pulling
`copy`	Copy image between registries/formats
`list-tags`	List all tags for a repository
`delete`	Delete an image from a registry
`sync`	Sync images between registries
`login`	Login to a container registry
`--raw`	Output raw manifest (with inspect)
`--override-arch`	Inspect specific architecture

Practical Examples

Example: Inspect image without pulling

```
$ skopeo inspect docker://docker.io/library/nginx:alpine
```

Get image metadata: layers, digest, labels, architecture, OS, and creation date - all without downloading the image.

Example: Copy between registries

```
$ skopeo copy docker://docker.io/library/nginx:alpine docker://ghcr.io/myorg/nginx:alpine
```

Copy an image directly between registries without pulling to local storage. Fast, efficient server-side transfer.

Example: List all tags

```
$ skopeo list-tags docker://docker.io/library/postgres | jq .Tags[]
```

List all available tags for a repository. Useful for finding version numbers and variants.

Example: Mirror images for air-gapped environment

```
$ skopeo copy docker://docker.io/library/nginx:alpine dir:/tmp/nginx-image/
```

Save image to a directory structure. Transfer the directory to air-gapped systems and load with skopeo copy dir: docker:// format.

Example: Check multi-arch support

```
$ skopeo inspect --raw docker://nginx:alpine | jq .manifests[].platform
```

View the manifest list to see which architectures (amd64, arm64, etc.) the image supports.

Tips & Best Practices

Pro Tip: No daemon required: Skopeo works without Docker or Podman daemons. Perfect for CI/CD pipelines and restricted environments where running a container daemon is not possible.

Note: Transport prefixes: docker:// = registry, dir: = local directory, oci: = OCI layout, docker-archive: = docker save tarball, containers-storage: = local Podman/CRI-O storage.

Pro Tip: Use with CI/CD: Skopeo is ideal for CI/CD image promotion: build ? push to staging registry ? test ? skopeo copy to production registry. No need to pull/push through the CI runner.

Note: Install skopeo: Install: apt install skopeo (Debian/Ubuntu 22.04+), dnf install skopeo (Fedora/RHEL). Available on all major distributions.