brewlet

CLI + Kubernetes cheat sheet

Ship the app. Not the runtime.

Brewlet 0.x | Pre-GA beta | linux/amd64 examples | A3 landscape

Your application is the workload
Code + dependencies under /app + explicit launch.json

+

The runtime is infrastructure
RuntimeImage approves a digest; NodeProfile offers a pool

→

Composed at container creation
containerd + Brewlet shim + runc | Separate app processes

01 Install + configure

Platform team | controlled node pools

Isolated Linux nodes: systemd, cgroup v2, containerd 2.x, runc and crictl. Provisioning is privileged. Current kube context assumed.

Install the latest CLI + control plane

```
curl -fsSL https://brewlet.sh/install.sh | sh
export PATH="$HOME/.local/bin:$PATH"
CHART=oci://ghcr.io/microsoft/charts/brewlet
helm show crds "$CHART" | kubectl apply -f -
helm upgrade --install brewlet "$CHART" \
  -n brewlet --create-namespace --wait
```

Unset `BREWLET_VERSION` if set. Keep the CLI, CRDs, chart and component images on the same release. No nodes are provisioned yet.

Approve a runtime first

```
brewlet k8s runtime add python-3.13 \
  --image registry.example.com/golden/python:3.13 \
  --platform linux/amd64 --dry-run
```

Review, then repeat without `--dry-run`. The CLI pins the digest and infers numeric non-root UID/GID; pass `--uid`/`--gid` if needed.

Offer it on a pool: node-profile.yaml

```
apiVersion: node.brewlet.sh/v1alpha1
kind: NodeProfile
metadata: {name: application-pool}
spec:
  nodePool:
    key: brewlet.example.com/pool
    names: [applications]
  runtimeImages:
    - {id: python-3.13, install: OnDemand}
  rollout:
    containerdRestart: validated
    maxUnavailable: 1
```

Use labels already on the intended nodes. Startup-fenced nodes also need worker tolerations: see the full installation recipe.

```
kubectl apply -f node-profile.yaml
kubectl wait --for=condition=Ready \
  nodeprofile/application-pool --timeout=5m
```

Eager (default)
Install roots before readiness.

OnDemand
Validate baseline; fetch at first use.

02 Build + deploy

Application team | ordinary Kubernetes

Describe the process: launch.json

```
{"schemaVersion":1,"process":{
  "entrypoint":["python3"],
  "args":["/app/src/main.py"],
  "workingDir":"/app"
}}
```

Explicit executable, arguments and platforms. No language detection or dependency resolution. Use an absolute executable if the runtime has no PATH.

Package, publish + exercise locally

```
APP=registry.example.com/apps/api:v2
brewlet build "$APP" --platform linux/amd64 \
  --launch launch.json --layer app=./src:/app/src
brewlet inspect "$APP"
brewlet push "$APP" --push-result push.json
brewlet run "$APP" --platform linux/amd64 \
  --runtime python-3.13 --from-cluster
```

`build` writes a local OCI layout; `push` publishes it. Add explicit dependency layers as needed. Local `run` uses Docker (or `--engine podman`), not cluster admission.

Select the runtime in your workload

Merge this fragment into `Deployment.spec.template`. Keep your labels, probes, resources and volumes.

```
metadata:
  annotations:
    brewlet.sh/runtime-image: python-3.13
spec:
  runtimeClassName: brewlet
  nodeSelector:
    kubernetes.io/arch: amd64
  containers:
    - name: api
      image: >-
        registry.example.com/apps/api@sha256:<digest>
```

Use the actual published digest and a runtime approved for your architecture. All application containers in a Pod use the same runtime ID.

```
kubectl -n apps apply -f deployment.yaml
kubectl -n apps rollout status deployment/api
```

Use an existing application namespace. Admission defaults the required read-only root to `true`; explicit `false` is rejected.

Direct delivery
build / push / apply

Governed delivery
Separate app + dependency approval.

03 Operate + rotate

Operations team | actual runtime bindings

Inspect the cluster

```
brewlet k8s runtime list
brewlet k8s runtime status python-3.13
brewlet k8s doctor --namespace brewlet
```

Preview a runtime change

```
brewlet k8s runtime rotate python-3.13 \
  --image registry.example.com/golden/python:3.13.1 \
  --dry-run
brewlet k8s runtime remove python-3.13 --dry-run
```

Review rotation, then repeat without `--dry-run` and add `--yes`. Application bytes are unchanged; running tasks are not hot-patched.

generation 1
Existing tasks retain their lease

→

generation 2
New tasks use the approved runtime

Compare approved + observed generations

```
brewlet k8s runtime status python-3.13 -o json \
  | jq '{generation, digest,
    taskObservations: .status.taskObservations}'
```

This is a snapshot, not an audit history. Check `current` and `observedAt`; stale observations can be retained.

Which generation was running at time T?

```
brewlet_container_runtime_image_info{
  namespace="apps", pod="api-<suffix>"
}
```

Run this as a Prometheus instant query at time T. First enable `metrics.enabled: true` and scrape node metrics. History is limited by retention and scrape gaps; short-lived containers can be missed. Historical lookup guide.

Govern dependencies separately

1 / `layer-set`	**2** / app-only `build`	**3** / `bundle`
4 / `verify`	**5** / test candidate	**6** / deploy digest

Compose immutable app + layer-set digests. Verify, test and deploy the same candidate digest. Runtime rotation does not patch dependencies packaged in the app.

More commands: `brewlet <command> --help`
Full CLI reference | Rotation + cleanup

Keep the boundaries explicit. /app is read-only and world-readable: never package secrets there. Use scoped volumes for data.
Digest-pin workloads. runc isolation, not a hostile-tenant sandbox. Evaluate pre-GA software on isolated infrastructure.

brewlet.sh/docs/cheat-sheet/
Complete recipes, prerequisites and safe removal online.
Copyright (c) Microsoft Corporation. MIT License.