Compose a kit set

- Choose the components
- Write the set descriptor
- Add runtime access to the set
- Build and publish the set
- Run the set
- Configure component arguments
  - Fix an argument's value
  - Let users choose a value
  - Change a build argument
- Review and update the composition
  - Check for file conflicts
  - Check combined settings
  - Update a component

{{< summary-bar feature_name="Docker Sandboxes sbx" >}}

A kit set gives your team one kit to run, with the tools and versions you've chosen. You list the kits to include, add any settings the combined environment needs, and publish the result. A set can also run setup commands, supply agent instructions, and offer choices such as which model to use.

Use a set when you want to share a combination of kits and manage its settings in one place. The components can be kits your team publishes or kits from other publishers. You need Docker Buildx and a registry namespace you can push to when publishing the set.

Choose the components

A set can contain one workload and any number of mixins. Together, they provide a complete sandbox environment. You can also make a set of only mixins, to share tools and settings that users add to a workload with `--kit`.

Choose v3 kits that work together. For example, you could combine an agent workload with a linter mixin and a mixin that adds your team's configuration. To package a tool of your own, see Build a tool mixin.

You can try a workload and mixins together with `sbx run` and `--kit` before composing a set. See Add mixins. If one kit depends on another, Docker Sandboxes applies the dependency first. Changing the order of `--kit` flags or entries in a set doesn't change that order.

Write the set descriptor

A set descriptor uses `kind: set` and lists its components under `kits:`. For example, this descriptor combines Docker's Codex workload with a mixin:

```
# syntax=docker/sandbox-kit:3
schemaVersion: "3"
kind: set
displayName: Codex with tools
version: "1.0.0"

kits:
  - ref: docker.io/docker/sbx-kit-codex:0.155.1
  - ref: <MIXIN_REFERENCE>
```

Replace `<MIXIN_REFERENCE>` with a published v3 mixin's full image reference. Choose a different workload reference to use another agent or environment. The components supply their own network rules and credential requests, so the set doesn't need to repeat them.

The set gets its software and files from the `kits:` list. It doesn't have a Dockerfile, a `dockerfile:` field, or a `build:` block. To include more software or static files, package them in a workload or mixin, publish that kit, and add it to the list.

Every `ref` must be a published registry reference. Local paths and Git URLs aren't accepted as components, even when building the set from local source.

Add runtime access to the set

You can give the combined environment access to services beyond those its components request. For example, to let the agent download packages from an internal registry, add this network capability to the set's descriptor:

```
capabilities:
  - type: com.docker.sandbox/network-policy@1
    config:
      runtime:
        allow:
          - packages.company.example:443
```

Replace the example domain with your registry's host. The rule permits access as long as the sandbox's policy allows it. If the registry also requires authentication, add a credential request. See Services declared by kits.

Keep access required by a tool in that tool's kit, so its access rules follow it when used with another workload. Put settings shared by the combined environment on the set, such as access to your team's package registry. You can also add setup commands, generate configuration files, provide instructions, and define arguments on the set.

Build and publish the set

Before publishing, check for file conflicts between the components.

Build and publish a set by passing its descriptor to Docker Buildx. For a descriptor at `codex-tools/codex-tools.yaml`:

```
$ docker login
$ docker buildx build ./codex-tools -f ./codex-tools/codex-tools.yaml \
    -t docker.io/<NAMESPACE>/codex-tools:1.0.0 --push
```

Replace the paths with your set's source directory and descriptor, and `<NAMESPACE>` with a Docker Hub namespace you can push to.

The build pulls the listed kits, checks their declared requirements, and combines their files and settings. It fails if a dependency is missing, two kits declare the same feature in `provides`, or more than one kit is a workload.

A set that includes a workload publishes as `kind: workload`. A set containing only mixins publishes as `kind: mixin`. Your team can use the result like any other workload or mixin. The published kit records each component's exact image digest in `kits:`. Only the source descriptor uses `kind: set`.

Run the set

Run a published set that contains a workload by passing its image reference to `sbx run`, as you would for an individual workload:

```
$ sbx run <SET_REFERENCE> --name my-project
```

The sandbox includes the workload and all the mixins packaged in the set. You don't need to list those mixins separately with `--kit`.

For a set containing only mixins, add it to a workload with `--kit`:

```
$ sbx run <WORKLOAD_REFERENCE> --kit <SET_REFERENCE> --name my-project
```

Authentication depends on the kits in the set. Check their documentation for required credentials, store those credentials on the host, and approve access when prompted. See Credential configuration for authentication options and preparing unattended runs.

To control the agent's base image or launch command, see Build an agent workload.

Configure component arguments

You can let your team change a setting without changing which kits are included. For example, they might need to choose whether a linter checks or fixes files while using the linter version you've selected.

A set controls which component arguments users can change. You can fix an argument's value when publishing the set, or expose it as an argument on the set so users can choose a value when creating a sandbox.

After changing the argument definitions in the descriptor, rebuild and publish the set.

Fix an argument's value

Suppose you have a linter kit with a `mode` argument that users normally set when creating a sandbox. Add it to the set with a fixed value of `check`:

```
kits:
  - ref: docker.io/my-org/linter-kit:1.0.0
    args:
      mode: check
```

This excerpt fixes the linter's `mode` to `check` when you publish the set. Use your linter's image reference and argument name. Other components in the set can have their own argument values.

Let users choose a value

To let users choose the mode when creating a sandbox, define an argument on the set and pass its value to the linter:

```
args:
  lint_mode:
    default: check
    enum: [check, fix]

kits:
  - ref: docker.io/my-org/linter-kit:1.0.0
    args:
      mode: ${{ kit.args.lint_mode }}
```

This example assumes the linter defines `mode` with the same default and allowed values. The set must preserve those constraints. Users can then pass `--kit-arg lint_mode=fix` when running the published set. They can change only arguments defined on the set, not other arguments on its components. See Set merge rules for the argument rules.

Change a build argument

Arguments used to build a component, such as the tool version to install, are fixed in its published image. To change one, rebuild and publish the component, then update its reference in the set.

Review and update the composition

Check for file conflicts

Before sharing a set, check that its components work together. Give each component's files separate paths: if two components include the same path, the file from the later image layer replaces the earlier one. The build orders layers by kit dependencies, so reordering `kits:` doesn't

choose which file wins. Inspect the built image for unintended replacements. When you combine kits with `--kit` instead, Docker Sandboxes rejects files that collide when it creates the sandbox.

Check combined settings

The set includes each component's network rules and agent instructions. Lifecycle hooks run in dependency order, but startup hooks run alongside the agent. The agent might start before those hooks finish.

Use install hooks for setup that must finish during sandbox creation. For setup that must finish before every agent launch, use the workload's entrypoint.

The workload chooses the instruction filename. If you add instructions with an `agent-context` capability on the set, leave out the filename.

Update a component

Change the component's reference, rebuild the set, and publish another version. Create another sandbox to try the updated set. Existing sandboxes keep the kit configuration they were created with.

A published set keeps the files and settings it was built with, even if a component's tag later points to a different image. To use those same images when rebuilding, add a `digest` beside each `ref` in the source.

For signing and multi-platform builds, see Build and distribute kits.

□□□ #2
□ □□□ □□□ 2026-10-31 19:00:02 UTC
□ □□□ □□□ 2026-10-31 19:00:04 UTC