


# Environments

Environments ensure your Ambient Agents run with consistent toolchains across all triggers. Learn when to use environments and how to configure them.

Environments ensure your [Ambient Agents](#) run with the same toolchain and setup every time, regardless of where they're triggered from. When you set up an environment, a workflow triggered from any surface uses identical dependencies, repos, and configuration, eliminating "works on my machine" drift.

An environment defines the execution context for automated agent runs: the **Docker image, repositories to clone, setup commands, and runtime configuration** Warp uses to prepare the workspace before the agent starts.

 You often don't need an environment for interactive local runs where you're already in a working checkout and relying on your existing machine setup.

## Key features

What environments give you:


- **Consistent behavior across triggers** – A workflow triggered from Slack behaves identically to one run from Linear or the CLI, using the same toolchain and setup steps every time.
- **One configuration, many uses** – Define your Docker image and setup once, then reuse it across triggers and hosts without duplicating configuration.
- **Full visibility into runs** – Inspect the image, repos, and commands used by a run, making it easy to debug failures or reproduce results.

## About environments

Environments define *how* an agent runs, not *what* it does. They're required for [Warp Platform](#) automation (Ambient Agents, integrations, API runs) but are not required for interactive local usage.

An environment typically includes:

- **Docker image (required)** – The toolchain and runtime the agent runs with.
  - The Docker image can be your own custom image, an official base image (e.g. node, python), or one of Warp’s prebuilt dev images (see [repo ↗](#)).
- **Repository/workspace** – One or more repos the agent can clone and operate on.
- **Setup commands** – Commands to prepare the workspace (e.g., dependency install, builds, bootstrapping).

 Configuring runtime settings:

- **Environment variables:** Configure these in your Dockerfile using Docker’s `ENV` directives or pass them when running the container.
- **Secrets:** For credentials and sensitive data, use [Agent Secrets](#). These are configured separately from environments and injected securely at runtime.

What an environment is not:


- **Host** – Hosts determine where execution happens (Warp-hosted vs. self-hosted infrastructure).
- [Agent Profile](#) – Profiles control agent behavior like permissions, model choice, and defaults, not the runtime environment.
- [Rules](#) – Rules determine agent responses and decisions but don’t define the container or toolchain.
- [MCP](#) – MCP connects agents to external tools and data. It doesn’t define or configure the execution environment.
- **Per-run context** – Trigger-specific data like Slack threads, PR metadata, or CI logs attach to individual tasks, not the environment configuration.

## How environments fit into the Warp Platform

An environment is the runtime layer for automated Warp Platform runs. It defines the container image, repos, and setup steps used when a trigger kicks off an agent task.

Components in the execution flow:

1. **Trigger** – An event starts work (Slack mention, Linear comment, CI event, API call)
2. **Task** – Warp creates a tracked task for the run
3. **Environment** – The task uses an environment to define execution context
4. **Host** – The environment runs on a host (Warp-hosted or self-hosted infrastructure).
5. **Agent execution** – The workflow runs in the prepared environment
6. **Outputs** – The run produces PRs, messages, reports, or transcripts

 **Local agent** runs (using `warp agent run`) don't require an environment. These runs use your current machine's setup. Environments are required for **automated platform** runs like Ambient Agents and integrations

## Hosts and environments

While environments define *how* an agent runs, hosts determine *where* the environment executes.

Host options:

- **Warp-hosted (default)** – Warp provides the infrastructure. Best for most users who want hands-off execution.
- **Self-hosted** – You provide the infrastructure (runners in your cloud or network). Best for compliance requirements, on-premise execution, or custom hardware needs.
- **Local (coming soon)** – Run environments on your local machine for sandbox development and testing.

The same environment can run on different hosts with identical behavior. For more details on hosting options, see [Deployment Patterns](#) and [Execution hosts](#).

## What happens at runtime

When you trigger an agent, Warp follows this process:

1. **Warp receives the trigger.** Warp captures the message content (Slack thread, Linear issue) and any linked context.

2. **Warp creates an execution environment.** Warp spins up an isolated execution context from the Docker image defined in your environment.
3. **Repositories are cloned.** GitHub repositories associated with the environment are cloned into the container.
4. **Setup commands run.** Configured setup commands execute (installing dependencies, running builds, etc.)
5. **The agent workflow runs.** The agent executes the task using the provided context, tools, and permissions.
6. **Results are posted back.** Progress updates, summaries, and results post to the trigger source (Slack, Linear, etc.), or are available in the task transcript..
7. **The container is destroyed.** After completion, the container is torn down. Each run starts from a clean, isolated environment.

This process ensures every run starts from the same baseline, making results reproducible and debugging straightforward.

## When to use environments

Use an environment when your run needs a predictable toolchain and repeatable setup, regardless of where it's triggered from.

- **Integrations and schedules** – Use an environment when runs start from Slack, Linear, GitHub Actions, schedules, or other integrations, and you need consistent behavior each time.
- **CI and remote automation** – Use an environment when the host isn't consistent (e.g., different runners, varying base images).
- **Team standardization** – Use an environment when you want everyone's automation runs to use the same image, repos, and setup steps.
- **Toolchain-specific workflows** – Use an environment when the workflow depends on specific language versions, linters, build tools, or system packages.

## When you can skip an environment

You often don't need an environment for interactive local runs where you're already in a working checkout and relying on your existing machine setup.

## Decision checklist

Choose an environment if any of the following apply:

- Runs must be consistent across triggers/hosts. The workflow should behave the same regardless of where it is triggered from.
- The toolchain must be fixed. You need a known image and deterministic setup steps to avoid “it works on my machine” drift.
- The workflow is shared across a team. Multiple people, or systems, will run the workflow and expect repeatable results.

### Example:

If your team tags @Warp in Slack to fix a failing CI job, an environment ensures every run uses the same Docker image, clones the same repos, and runs the same setup commands.

The fix the agent tests matches what runs in CI and what your teammates see when they review the PR.

## Where to configure environments


You can create and configure environments with Warp’s guided setup, or through the CLI. Use the guided flow when you’re first getting started, and use the CLI when you want full control or need to automate environment creation.

### Before you begin

Make sure you have:

- One or more GitHub repositories that the agent should clone and work in.
- A publicly-accessible Docker image that can build and run your code. Official images like [node ↗](#), [python ↗](#), or [rust ↗](#) work for many projects.
  - **Optional:** use Warp’s prebuilt dev images

- If you don't have a custom Docker image yet, Warp maintains a small set of prebuilt Ubuntu-based development images (Node/Python base, plus Go/Rust/Java/.NET/Ruby variants) that work well for common repos.
- You can use them as a starting point, or bring your own image. Warp only uses these if you explicitly choose them or if you don't supply a `--docker-image`.
- See: [Warp Dev Environment Docker Images ↗](#)

 Create one environment per codebase, then reuse it across triggers like Slack, Linear, and CLI runs.

## Create an environment with guided setup (recommended)

Use `/create-environment` ↗ when you want Warp to inspect your repos and recommend an environment configuration automatically. This is the fastest way to get started: Warp detects your languages, frameworks, and tools, then suggests appropriate images and setup commands.

You can run the command inside a git repo directory with no argument, or with one or more repo paths or URLs.

```
# Local file paths
/create-environment ./warp-internal ./warp-server

# owner/repo
/create-environment warpdotdev/warp-internal warpdotdev/warp-server

# GitHub URLs
/create-environment
https://github.com/warpdotdev/warp-internal.git
```

Warp will:

- Detect the repositories you want the agent to work with and identify languages, frameworks, and tools
- Look for an existing Dockerfile, recommend an official base image, or help build a custom image (if needed)

- If you don't provide an image, Warp can suggest an official base image or a [Warp prebuilt dev image ↗](#) as a starting point.
- Suggest setup commands based on your scripts and package managers
- Create the environment through the CLI and return an `environment ID`

This produces a ready-to-use environment that can immediately be connected to integrations and Ambient Agents.

## Create an environment with the CLI

Use the CLI when you already know how you want to configure your environment, you have a custom Docker image you want to use, or when you're automating environment creation.

```
warp environment create \
  --name <name> \
  --docker-image <image> \
  --repo <owner/repo> \
  --repo <owner/repo> \
  --setup-command "<command1>" \
  --setup-command "<command2>"
```

Required flags:

- `--repo` : repo to clone (repeatable).
- `--setup-command` : commands run in the order provided (repeatable).

Optional flags:

- `--name` : human-readable label for the environment.
- `--docker-image` : image name on Docker Hub.
  - If you pass `--docker-image`, Warp will use it exactly. If you don't, Warp may suggest a starting image during setup, which you can always override. Also choose from a set of preset images [here ↗](#).

## Managing environments

Once created, you can use the [Warp CLI](#) to inspect and update environments.

## List environments

```
warp environment list
```

**View an environment's configuration.** Replace <ENV\_ID> with the ID of the environment you want to view.

```
warp environment get <ENV_ID>
```

## Update an environment

Add/remove repos and setup commands without recreating the environment. Replace <ENV\_ID> with the ID of the environment you want to modify.

```
# Add a repo
warp environment update <ENV_ID> --repo owner/repo

# Remove a repo
warp environment update <ENV_ID> --remove-repo owner/repo

# Add a setup command
warp environment update <ENV_ID> --setup-command "your command"

# Remove a setup command (must match exactly)
warp environment update <ENV_ID> --remove-setup-command "exact command"
```

**Delete an environment.** Replace <ENV\_ID> with the ID of the environment you want to delete.

```
warp environment delete <ENV_ID>
```

 For end-to-end setup, see the [Integrations and Environments](#) guide.



## Environment design and best practices

A well-designed environment removes guesswork by giving every run the same starting conditions. When an agent opens a PR from Slack or fixes a failed CI job, the result matches what your team can reproduce locally and in CI.


### Design guidelines

- **Keep setup repeatable** – Write setup steps that are safe to rerun and that produce the same toolchain and workspace state for a given repo revision. This keeps agent runs reliable across triggers and hosts.
- **Pin versions in the toolchain** – Prefer a Docker or base image that pins language runtimes and core tools, then use lockfiles (`package-lock.json`, etc.) for dependencies.
- **Define a clear workspace boundary** – In multi-repo environments, explicitly state which repos are cloned and where setup commands run so the agent doesn't "guess" the working directory.
- **Make prerequisites explicit** – If the agent must run a build step, generate code, or install system packages before it can do meaningful work, encode that as setup.

### Example setup commands

```
# Safer patterns (repeatable and stable)
mkdir -p .cache
npm ci

# Less safe patterns (can fail on rerun or drift over time)
mkdir .cache
npm install
```

 If your setup commands depend on secrets or credentials, configure them through Warp's [secrets mechanism](#) rather than hardcoding tokens. This keeps credentials secure and makes runs fully automated.

### Common issues

- **Setup assumes previous state** – Steps that rely on leftover caches, existing directories, or already-cloned repos can make runs unreliable.
  - Solution: Write idempotent setup commands that work on a fresh container.
- **Missing credentials or secrets** – Builds fail when private repos, package registries, or external services require authorization.
  - Solution: Configure credentials with [Agent Secrets](#).
- **Repo access and GitHub authorization issues** – Runs fail when GitHub doesn't have repo access or the triggering user lacks permissions.
  - Solution: See [How GitHub Authorization works](#) for Github authorization setup.