BuilderStudio

Comprehensive User Guide

A practical guide to building, automating, previewing, and shipping software with BuilderStudio by WunderCorp.

What BuilderStudio is for

BuilderStudio is a local-first agentic development workspace. It brings together project files, AI models, skills, pathways, MCP tools, native connectors, workflow design, local previews, deployment handoff, and optional contained Hermes execution.

```
Workspace -> Model profile -> Skills / Pathways / MCP / Connectors
          -> Approval gate -> BuilderStudio daemon execution
          -> Optional explicit /hermes contained execution
          -> Changed files, diagnostics, preview, package, or deployment handoff
```

How to use this guide

Read the quick start first, then jump to the feature area you are configuring or explaining to a teammate.

Section	What it covers
Quick start	First run, model setup, workspace import, agent prompt, preview
Workbench	Files, editor, search, terminal, project detection, previews
Models	BYOK profiles, OpenRouter, OpenAI, DeepSeek, Ollama, LM Studio
Agentic Development	Conversation mode, plan mode, build runs, approvals, right-side canvases
Skills and Pathways	Reusable instructions and guided starter build recipes
MCP and Connectors	Tool servers, desktop apps, SaaS connectors, approval policy
Workflows	Visual workflow builder, triggers, actions, gates, templates
Runtime and Deployment	Local Apps, auto-heal, ShipYou handoff, Pro Mode
Hermes and Swarms	Container-only agent execution and parallel lane runs
Security and Ops	Trust boundary, settings, commands, troubleshooting

The core loop

- Open or create a trusted workspace.
- Choose a model profile, including BYOK or local runtime options.
- Attach skills, pathways, MCP tools, or connectors when needed.
- Choose /conversation, /plan, or /build based on intent.
- Approve risky actions before writes, commands, installs, or runtimes run.
- Review changed files, diagnostics, commands, previews, and deployment output.

Quick start

A typical first session takes you from setup to a running local app.

1. Open the app

Launch BuilderStudio and sign in if you plan to use ShipYou, Pro Mode, organizations, or deployment handoff.

2. Choose a model

Open Key Management or Integrations. Save OpenRouter, OpenAI, DeepSeek, Ollama, LM Studio, or a compatible profile.

3. Open a workspace

Use the Builder workbench to create, open, import, or browse a local project directory.

4. Ask the agent

Go to Agentic Development. Use /conversation to discuss, /plan to outline, or /build when you want edits and execution.

5. Approve actions

Review capability requests before writes, terminal commands, installs, runtime starts, MCP tools, or deployment actions.

6. Review output

Inspect changed files, terminal output, diagnostics, runtime preview, and any packaged or deployed artifacts.

Useful first commands

```
/demo          guided product tour
/setup         prepare the machine for local development
/conversation  talk with the AI without building or editing
/plan          draft an implementation plan without running a build
/build         return to normal build/edit/run behavior
/pathways      open starter prompt cards
/helloworld    create a tiny first app
/hermes        open contained Hermes setup
```

Product map

BuilderStudio is organized around local project work, model setup, agent execution, tools, deployment, and account settings.

Area	Use it for
Builder	File tree, editor, preview, terminal, local project operations
Agentic Development	AI plans, code edits, runs, Hermes, swarms, workflow mode
Key Management	Saved AI provider profiles and secret lifecycle
Integrations	Guided AI and provider setup
Native Connectors	Connect agents to desktop and SaaS apps
MCP Control Center	Register and inspect MCP tool servers
Deployments	Preflight, package, upload, and ShipYou handoff
Local Apps	Track running previews and local ports
Settings	Runtime provider, appearance, defaults, safety reminders
Profile / Organizations / Pricing	Account identity, organization context, subscription, billing, and referral surfaces

Desktop architecture

The renderer owns the interface. The BuilderStudio daemon owns sensitive local operations: filesystem access, terminal execution, model requests, provider secrets, local runtimes, MCP tool access, source packaging, native connector actions, and ShipYou deployment handoff.

Builder workbench

The Builder route is the local project surface for files, editing, preview, terminal work, and workspace operations.

Workspace operations

Create a workspace, open an existing folder, import a browser directory, reveal the folder on disk, or return to recent workspaces.

Project detection

BuilderStudio inspects package.json, scripts, Dockerfiles, dependencies, dev dependencies, package manager signals, and framework hints.

File tree and editor

Browse directories, open files, create files, rename paths, delete paths, and work with browser-imported or local filesystem projects.

Search and replace

Search project files and replace text across browser-imported workspaces when appropriate.

How to use it

- Open the Builder route when you want to inspect project structure before asking the agent to edit.
- Use project detection to understand scripts, dependencies, and preview options.
- Use the editor for direct inspection and small manual changes.
- Use reveal/open-folder actions when you need to move between BuilderStudio and your operating system.

Terminal, preview, and runtime canvas

BuilderStudio connects local commands to preview surfaces so generated or existing apps can be run and inspected.

Terminal execution

Run approved terminal commands through the local daemon. Use this for package scripts, checks, tests, install steps, and one-off commands.

Runtime start and stop

Start and stop local apps through runtime routes. BuilderStudio can track runtime URL, command, status, port, and provider.

Local Apps panel

View running and recent preview ports. Stop stale servers, open URLs, and manage local app history.

Runtime providers

Choose auto, Docker Desktop, Colima, or host Node for generated apps depending on local setup and sandboxing needs.

```
Runtime provider preference:
auto -> Docker Desktop -> Colima
docker-desktop -> Docker Desktop
colima -> Colima with Docker runtime
host-node -> local Node quarantine
```

Models, BYOK, and OpenRouter Fusion

BuilderStudio keeps provider choice explicit. The daemon calls the selected provider with user-owned keys or local endpoints, and Fusion is an opt-in OpenRouter workflow.

OpenRouter

Save one OpenRouter key, refresh models, select exact model IDs, and keep strict routing enabled when provider fallback is not wanted.

OpenRouter Fusion

Fusion keeps the selected outer OpenRouter model and adds an optional analysis panel of 1-8 models, with an optional judge and controlled web calls.

Direct and compatible APIs

Use OpenAI direct, DeepSeek direct, or a custom OpenAI-compatible endpoint by defining the base URL, model ID, and key requirements.

Local models

Use Ollama, LM Studio, or another compatible local endpoint. Local profiles avoid cloud model traffic for supported tasks.

Provider and Fusion settings

```
profileId  displayName  providerKind  baseUrl  model  local  enabled
apiKeyPresent  apiKeyEnvironmentVariable  defaultTemperature  defaultMaxTokens
fusion.enabled  fusion.mode: auto | always
fusion.analysisModels[1..8]  fusion.judgeModel?  fusion.webCalls[1..16]
```

Local models

Local runtimes let BuilderStudio run compatible agent and chat flows without a cloud model provider.

Ollama

Use a local Ollama model through its OpenAI-compatible endpoint. Default endpoint: http://127.0.0.1:11434/v1.

LM Studio

Use a loaded LM Studio model in server mode. Default endpoint: http://127.0.0.1:1234/v1.

Container note

When a local model is used from inside Hermes or another container, localhost may need to be translated to a host-accessible address such as host.docker.internal depending on the runtime provider.

When to use local models

- You want local-only inference for compatible tasks.
- You want to avoid per-token cloud provider cost.
- You are testing lightweight planning or editing workflows.
- You are running in an environment where cloud providers are unavailable.

Agentic Development

The Agentic Development route is where BuilderStudio chats, plans, edits, runs, previews, packages, and reviews AI-assisted work.

Prompt and plan

Start in /conversation for discovery, /plan for a no-build implementation plan, or /build when edits and execution are intended.

Run payload

Runs include the selected workspace, prompt, provider profile, exact model, skills, pathway context, workflow context, approvals, and runtime settings.

Right-side canvases

Normal visual generation stays on the simulator/runtime canvas. Output is selected automatically only for genuinely backend-only work where no visual preview applies.

Review output

Inspect the summary, changed files, terminal commands, diagnostics, transcript, runtime URL, generated app path, and validation evidence.

Conversation, Plan, and Build modes

- /conversation - discuss ideas, requirements, alternatives, and tradeoffs without writing files or starting runtimes.
- /plan - produce a structured plan, risks, dependencies, acceptance criteria, and validation checklist without execution.
- /build - enter the approval-aware daemon workflow for edits, checks, previews, packaging, or deployment handoff.

Capabilities and approvals

BuilderStudio treats local execution as a permissioned workflow, not a silent background action.

Capability	Purpose
read-workspace	Inspect files, metadata, scripts, and editor state
search-project	Search source files and build a project map
write-files	Create and update trusted workspace files
delete-files	Remove files during cleanup or refactors
run-terminal	Run scripts, checks, tests, linters, and commands
install-dependencies	Install or update packages
start-local-runtime	Start local apps, APIs, or dev servers
manage-skills	Install and attach local SKILL.md bundles
use-mcp-tools	Call enabled MCP tools through local daemon policy
read-env-references	Read env variable names and references without exposing values
package-source	Create deployment source bundles
deploy-preview	Start final approved deployment handoff

Automation modes

- manual - the user keeps tight control over the run.
- approve-edits - the agent can prepare work, but the user approves changes.
- autonomous-local - broader local automation, still bounded by workspace trust and dangerous capability approvals.

Skills

Skills are reusable instruction bundles that change how the agent behaves for specialized work.

What a skill contains

A skill can include a SKILL.md file, reference files, activation keywords, install path, source, and metadata.

Skill context pack

BuilderStudio can build a bounded context pack with included files, skipped files, prompt prefix, skill names, and character count.

Skill discovery

The app can discover public GitHub repositories that look like skills and infer metadata from manifests, repo names, topics, and descriptions.

Approval-aware install

Skill installs should go through a generated command, user approval, local terminal executor, and a refreshed skills list.

Use skills when

- The task needs a repeatable standard or style guide.
- A domain has special rules the model should follow.
- You want a prompt shortcut with reference files attached.
- You want to reuse the same instructions across projects.

Pathways

Pathways are guided starter flows. They help a user choose what to build, then prefill the agent with a structured prompt and matching skills.

How pathways differ from skills

Skills tell the agent how to behave. Pathways guide the user through what to build or how to start.

How to open

Type /pathways or use the command palette. Choose a starter card, review the prompt, adjust details, and run it through Agentic Development.

Included pathway examples

- 3D creator portfolio, cinematic space travel page, responsive hero, SaaS landing page.
- Conversion CTA, dark email capture, DeFi dashboard hero, DesignPro hero.
- Liquid glass hero and footer, looping video hero, cinematic video hero.
- NFT/Orbis page, Prisma studio, secure SaaS hero, Aura and Axion inspired pages.

Best practice

Use pathways to start faster, then edit the generated prompt before running. This keeps the build specific to your product instead of blindly accepting a generic starter.

MCP Control Center

MCP support lets BuilderStudio register tool servers and expose approved capabilities to agents through the local daemon.

Server registry

List registered MCP servers through /mcp/list. Each server includes ID, display name, transport, command or URL, enabled state, and creation time.

Register servers

Register stdio or streamable HTTP servers through /mcp/register. Use stdio for local tools and streamable HTTP for hosted tool endpoints.

Security model

MCP tools are sensitive. Agents should call enabled MCP tools through the local daemon policy layer, not by bypassing approvals from the renderer.

```
McpServerDescriptor:
serverId
displayName
transport: stdio | streamable-http
command? args? url?
enabled
createdAt
```

Native Connectors

Native Connectors let agents work with desktop apps, developer tools, creative tools, and SaaS services through explicit connector contracts.

Connector catalog

Current connector targets include Blender, Spline, Figma, FL Studio, Ableton Live, Unity, Unreal Engine, VS Code, Terminal, Docker Desktop, GitHub Desktop / Git, Adobe Photoshop, Adobe After Effects, DaVinci Resolve, Postman, Supabase, Notion, Slack, Google Workspace, and OBS Studio.

Categories

Connectors are grouped by 3d, design, music, video, game, dev, productivity, ops, and data workflows.

How execution works

The UI owns discovery, model-routing labels, setup notes, risk labels, and approvals. The daemon or companion host owns OS-level access.

When to use connectors

- A task needs to control or inspect an external app.
- The agent needs to create assets in design, 3D, music, video, game, or development tools.
- A SaaS service needs approved agent actions such as posting, syncing, or querying data.

Workflow Builder

The visual workflow designer helps users build automations with triggers, actions, logic, agent steps, and approval gates.

Canvas and panels

Use the zoomable canvas, templates, inspector, contract panel, and generated prompt handoff to shape a workflow.

Node types

Use triggers, actions, logic gates, data steps, agent steps, approval steps, deployment steps, and response steps.

Available building blocks

- Triggers: webhook, schedule, manual, Gmail, GitHub, Salesforce, X content.
- Actions: HTTP request, edit fields, code/function, send email, Salesforce, GitHub repo source, GitHub repo create/update, Read PDF, ShipYou deploy, respond to webhook.
- Logic: If, Switch, Merge, Wait, Filter Gate, AND Gate, OR Gate, Error Boundary, Human Approval.
- Agent: Agent Draft, Kaize X Post, content workflow steps.

Templates

Templates include Kaize X content engine, Salesforce lead routing, GitHub repo to ShipYou deploy, document review operations, and scheduled API sync.

GitHub and source workflows

BuilderStudio includes GitHub setup and import surfaces for repository-backed development and deployment workflows.

GitHub OAuth

Connect or disconnect GitHub through BuilderStudio cloud routes. Once connected, list repositories available to the user.

Repository import

Import a repository through the daemon, clone it locally, and open it as a BuilderStudio workspace for agent work.

Git canvas

Agentic Development includes a Git-oriented right-side canvas for repository configuration, diffs, conflict handling, and push flows.

Conflict assistance

Commands such as /diff and /autoresolve support conflict analysis and LLM-assisted resolution preparation.

```
GitHub routes:
/integrations/github/status
/integrations/github/oauth/start
/integrations/github/repositories
/integrations/github/disconnect
/github/import
```

Deployment Center and ShipYou handoff

Deployment support prepares local source for ShipYou without making the renderer own packaging or cloud secrets.

Deployment preflight

Inspect project shape before upload. Report issues, inspected file count, and project detection metadata.

Source packaging

The daemon creates sanitized source bundles while respecting ignore rules, checksums, archive creation, and size limits.

Staging upload

Upload packaged source to ShipYou staging after local preparation and authorization.

Deployment handoff

Create a handoff to ShipYou with app name, slug, organization context, deployment prompt, and staged source.

```
Deployment routes:
/deployment/preflight
/source/pack
/shipyou/staging-upload
/shipyou/deployment-handoff
```

BuilderStudio Pro Mode

Pro Mode uses ShipYou credits to generate complete source bundles and import them into the local desktop workspace.

Credit-based generation

Sign into ShipYou, check Builder credit balance, estimate cost, and start a source-bundle generation job.

Job lifecycle

Start jobs, poll status, cancel active jobs, download generated zips, and track generation progress.

Import and run

After a Pro source zip is generated, BuilderStudio can import the zip through the desktop daemon, open it as a workspace, and optionally run it locally.

Good use cases

- Generating a complete app skeleton from a detailed product brief.
- Creating a polished landing page, dashboard preview, or auth-ready starter.
- Producing source that a user wants to inspect, modify, and run locally.

Hermes contained execution

Hermes is an optional contained runner. It is never the implicit runtime for ordinary builds, orchestration, saved preferences, or swarm fallback.

Command-only launch

Hermes can start only from the /hermes command path. The run must carry the explicit Hermes invocation marker from the UI to the daemon.

Daemon enforcement

The daemon independently rejects Hermes requests that did not originate from /hermes, including forged preferHermes flags or stale runtime preferences.

Container-only posture

```
executionIsolation: container
hermesExecutionMode: container-only
workspaceMountMode: selected-workspace-only
hostShellAccessAllowed: false
hostInstallAllowed: false
mountHostHomeDirectory: false
mountHostSshDirectory: false
secretsMode: explicit-env-only
```

How to use it

- Use /hermes status, /hermes install, and /hermes setup to prepare the contained runner.
- Use /hermes run <request> when the user deliberately wants Hermes against the active project.
- Use /build for normal daemon execution. Enabling orchestration or a swarm does not enable Hermes.

Agentic Swarming

Swarming turns one goal into independently reviewable lanes while preserving workspace trust, capability approvals, and explicit runtime selection.

Commands

Use /swarm, /swarm list, /swarm demo, /swarm launch-stack, or /swarm security-audit from Agentic Development.

Scenario model

A scenario defines an ID, title, description, recommended use, default goal, and lane descriptors with bounded objectives and outputs.

Execution model

The daemon creates a session directory, runs lanes through the selected supported runtime, and aggregates results. Swarm fallback cannot silently launch Hermes.

Review model

Each lane has an ID, role, objective, prompt, capabilities, output directory, diagnostics, and evidence that can be reviewed before integration.

Runtime rule

- A normal /swarm command uses the configured daemon or selected compatible runtime.
- Hermes remains separate and requires an explicit /hermes invocation.
- All lane writes remain inside the selected workspace and approved mutation envelope.

Security and trust model

BuilderStudio separates user interface, local daemon, provider secrets, tool access, and execution approvals.

Daemon boundary

Sensitive operations stay behind the local daemon: files, terminals, providers, runtimes, packaging, connectors, MCP, and ShipYou handoff.

Workspace trust

Writes, deletes, terminal commands, installs, runtime starts, MCP tools, source packaging, and deployments require appropriate trust and approval.

Secrets

Provider keys should be saved locally by the daemon. The renderer receives profile metadata, not raw secrets after setup.

Local-first AI traffic

Agent and code traffic is intended to use desktop-direct BYOK or local runtimes, with cloud API fallback disabled for those flows.

```
llmRuntimeMode: desktop-direct-byok-or-local
cloudApiFallbackAllowed: false
```

Settings, account, and billing

BuilderStudio includes account surfaces for ShipYou integration plus local preferences for runtime, appearance, and safety.

Settings

Control appearance, generated app container runtime provider, default workspace folder, default deployment region, agent approval mode, and shortcuts.

Profile and login

Sign in for account identity, ShipYou links, deployment handoff, Pro credits, organizations, and subscription management.

Organizations

Select organization context for deployment handoff and account-scoped work.

Pricing and referral

Access pricing, subscription, checkout, billing portal, and referral program surfaces.

Safety reminders in settings

- Workspace trust is required for writes.
- Terminal commands stay local.
- MCP tools need approval policy.
- Deployment apply requires a signed-in user.

Command palette and slash commands

Commands provide shortcuts into setup, workflows, runtime views, deployment, and agent modes.

Core commands

/demo, /start, /getting-started, /pathways, /helloworld, /setup, /conversation, /plan, /build, /hermes, /hermes install, /hermes setup, /swarm, /swarm list, /swarm demo, /swarm security-audit.

Agent surface commands

/workflow, /simulator, /terminal, /git, /diff, /autoresolve, /push, /deploy, /colima, /docker, /reset, /clear, /referral.

How to think about commands

Commands are not separate products. They are shortcuts into BuilderStudio routes and agent modes. Use them when you know the destination and want to skip manual navigation.

Agent prompt modes

```
/conversation  Open a build-free conversation lane for questions, discovery,
               ideation, and clarification.
/plan          Ask for a structured plan, milestones, risks, dependencies,
               and validation notes without editing files.
/build         Switch back to the normal agentic builder behavior for edits,
               commands, previews, packaging, and deployment.
```

Troubleshooting and recovery

Use these checks when local runs, agent output, model calls, or deployment handoff do not behave as expected.

Model is not responding

Check provider profile status, key presence, base URL, selected model ID, local runtime server state, and strict routing settings.

Preview is blank

Check runtime URL, port state, build logs, package install status, Vite start output, and auto-heal diagnostics.

Agent cannot write

Confirm workspace trust, selected capabilities, permission approval, and whether the app is running in browser-imported or local daemon-backed mode.

Deployment blocked

Run preflight, resolve errors, confirm sign-in, select organization, package source, stage upload, then retry handoff.

Auto-heal scope

Runtime auto-healing is deterministic and bounded. It can repair known problems such as dependency drift, wrong package imports, Tailwind/Vite wiring, missing safe dependencies, and stale preview-port detection. It should not perform open-ended autonomous repair without review.

Feature reference matrix

Use this matrix to quickly identify where a feature lives and whether it depends on the desktop daemon.

Feature	Primary route	Requires daemon?
File workbench	Builder	Yes for local filesystem
Agent runs	Agentic Development	Yes
Model provider profiles	Keys / Integrations	Yes
Skills	Agentic Development	Yes for local skill list/context
Pathways	Agentic Development	No for prompt cards; yes for execution
MCP servers	MCP Control Center	Yes
Native connectors	Native Connectors	Yes for execution
Workflow builder	Workflows / Agent mode	Partly
Local previews	Builder / Agent / Local Apps	Yes
Deployment handoff	Deployments	Yes
Pro Mode	BuilderStudio Pro	Yes plus ShipYou account
Hermes	Agentic Development	Yes
Swarming	Agentic Development	Yes

Practical rule

If a feature touches local files, secrets, commands, runtimes, source packages, external apps, MCP servers, or deployment APIs, assume the daemon is the source of truth.

Glossary and final checklist

A compact reference for common BuilderStudio terms and the recommended workflow before shipping work.

Term	Meaning
Workspace	The selected trusted local project root and hard write boundary
Model profile	Saved provider configuration for cloud, BYOK, or local model calls
Skill	Reusable instruction bundle and optional reference context
Pathway	Guided starter prompt flow for common build patterns
MCP server	Tool server registered for agent use through local policy
Native connector	Approved bridge to a desktop or SaaS app
Hermes	Optional contained runner launched only through /hermes
Swarm	Parallel multi-lane run using the selected supported runtime; no implicit Hermes fallback
Fusion	Opt-in OpenRouter multi-model deliberation attached to the selected outer model
ShipYou handoff	Packaging and staged deployment transfer to ShipYou

Before you ship

- Confirm the selected workspace is the intended project and all generated targets remain inside it.
- Confirm the provider profile, exact model, and Fusion state are the settings you intended to use.
- Review capabilities before approving writes, commands, installs, tools, connectors, runtimes, or deployment.
- Inspect changed files, validation output, diagnostics, and the simulator preview.
- Run deployment preflight before packaging and ShipYou handoff.

Appendix: route reference

Developer-facing reference of the daemon and cloud route families surfaced in the source bundle.

```
/ai/providers/*
/ai/chat
/agent/session
/agent/plan
/agent/run
/agent/scripts/draft
/skills/list
/skills/context
/mcp/list
/mcp/register
/workspace/list
/workspace/open
/workspace/create
/terminal/run
/runtime/start
/runtime/list
/runtime/stop
/deployment/preflight
/source/pack
/shipyou/staging-upload
/shipyou/deployment-handoff
/hermes/status
/hermes/install
/hermes/setup
/swarm/scenarios
/swarm/run
```

Aurelius Agent

Aurelius Agent is BuilderStudio's built-in orchestrator for larger product, workflow, security, research, and multi-agent work.

What Aurelius Agent is

Aurelius plans complex work, creates scoped lanes, reviews results, and decides whether to integrate, repair, retry, or ask for approval. It does not implicitly launch Hermes.

Internal identity

Runtime ID: aurelius-agent. Display name: Aurelius Agent. Public product name may be WunderCorp Aurelius. Execution remains subject to daemon policy.

Default role

Use Aurelius for broad planning, orchestration, review, integration, and bounded multi-lane work. Hermes is a separate explicit command-only runner.

Runtime identity

```
runtimeId: aurelius-agent
displayName: Aurelius Agent
role: orchestrator
containerImage: ghcr.io/wundercorp/aurelius-agent:latest
mayDelegateToHermes: false
secretsMode: explicit-env-only
```

Aurelius, Hermes, and the daemon

The three components have separate roles. The daemon enforces policy, Aurelius orchestrates when selected, and Hermes runs only after an explicit /hermes command.

Aurelius Agent

Orchestrator, planner, workflow manager, security reviewer, integrator, and final decision layer for complex work.

Hermes Agent

Optional contained coding runner. It receives work only through the explicit /hermes command path.

BuilderStudio daemon

Secure execution layer that owns workspaces, mounts, secrets, capabilities, approvals, logs, and runtime lifecycle.

Execution flows

```
/build or /aurelius run -> daemon policy -> selected runtime -> validation -> report
/hermes run <request>  -> explicit marker -> daemon policy -> Hermes container -> report
```

Rules of thumb

- Aurelius does not delegate to Hermes automatically.
- Saved runtime settings and preferHermes flags do not make Hermes eligible.
- All execution routes remain subject to workspace boundaries, approvals, and audit logs.

Secure container execution

Aurelius Agent is designed to run as a containerized, policy-controlled runtime with narrow workspace access.

Secure defaults

Aurelius Agent runs container-first, with target-workspace-only mounts, explicit environment secrets, restricted host access, and approval gates for external side effects.

Private source, runnable image

WunderCorp can keep Aurelius source private while BuilderStudio pulls and runs the published image from GHCR.

Default posture

```
executionMode: container
networkPolicy: none by default
workspaceMountMode: target-workspace-only
requiresHostHomeMount: false
requiresHostSshMount: false
requiresHostShellAccess: false
mountDockerSocket: false
secretsMode: explicit-env-only
```

Actions that require approval

- external network access beyond allowed model or package registry paths
- sending messages, posting social content, or updating customer systems
- production deployment, cloud write operations, or infrastructure changes
- access to secrets, provider keys, SaaS connectors, or native desktop connectors

Using Aurelius Agent

Aurelius is a built-in runtime option. Ordinary builds remain on BuilderStudio daemon execution unless the user explicitly selects another supported route.

Built-in runtime

BuilderStudio can seed Aurelius in the daemon runtime registry. Manual runtime registration remains for third-party or team-provided runtimes.

Recommended runtime selector labels

Option	Use it for
BuilderStudio daemon	Default build, edit, command, validation, preview, and packaging workflow
Aurelius Agent	Planned product, workflow, security, research, operations, or multi-lane orchestration
Manual runtime	Third-party or custom runtimes registered by advanced users
Hermes	Not a general selector default; start only with /hermes

Useful commands

- /aurelius run <request> - run Aurelius against the selected workspace.
- /swarm - start a multi-lane flow without implicit Hermes fallback.
- /hermes - open Hermes status and run Hermes only when explicitly requested.
- /runtimes - inspect built-in and manual runtimes when exposed by the UI.

Manual runtimes

Manual runtime registration should remain available for advanced users, but Aurelius Agent should not depend on it.

Manual registration purpose

Use manual registration when a user wants to bring a custom runtime, internal research agent, team-specific worker, or third-party container into BuilderStudio under daemon policy.

Built-in IDs are reserved

- aurelius-agent is reserved for the first-party Aurelius Agent runtime.
- hermes-agent is reserved for the first-party Hermes Agent runtime.
- Manual registration must not override BuilderStudio internal daemon execution.
- Manual registration should reject attempts to override built-in runtime IDs.

Manual runtime payload shape

```
{
"runtimeId": "team-custom-agent",
"displayName": "Team Custom Agent",
"distributionMode": "container",
"executionMode": "container",
"containerImage": "ghcr.io/example/team-custom-agent:latest",
"entrypoint": "node dist/index.js",
"secretsMode": "explicit-env-only"
}
```

Aurelius Agent troubleshooting

Use these checks when Aurelius is not visible, cannot pull its image, or behaves differently from the selected runtime contract.

Runtime not visible

Confirm the daemon seeded built-in runtimes and the UI reads /agent/runtimes. The display name should be Aurelius Agent.

Image pull denied

Authenticate the container runtime to the registry with an appropriate package-read credential. Do not expose that credential in logs or screenshots.

Unexpected Hermes launch

Treat this as a routing defect. Aurelius, /build, orchestration, and swarm fallback must not launch Hermes without an explicit /hermes marker.

Hermes request rejected

This is expected when a request lacks the /hermes invocation source. Start Hermes from /hermes run instead of forcing runtime flags.

Quick checks

```
docker pull ghcr.io/wundercorp/aurelius-agent:latest
docker run --rm ghcr.io/wundercorp/aurelius-agent:latest describe
docker run --rm ghcr.io/wundercorp/aurelius-agent:latest smoke
curl -s http://127.0.0.1:48741/agent/runtimes | jq
```

Cross-platform IDE and remote runtimes

BuilderStudio can now run as a local desktop client or connect to a Hermes daemon on a VPS or through an SSH tunnel.

Local desktop

The existing macOS default remains unchanged. A local daemon still runs on 127.0.0.1 and owns files, terminal commands, previews, packaging, provider keys, and connector execution.

Remote VPS or SSH tunnel

Linux and Windows clients can connect to a remote Hermes runtime target. SSH tunnel mode keeps the daemon private while exposing a local forwarded URL to the IDE.

Runtime target modes

Mode	Use it for	Typical endpoint
Local desktop	Mac, Linux, or Windows app talking to its bundled local daemon.	http://127.0.0.1:48741
Remote VPS	Browser or desktop IDE talking to a daemon exposed behind HTTPS and token auth.	https://vps.example.com/hermes
SSH tunnel	Private VPS daemon forwarded to the local machine without opening the daemon to the internet.	http://127.0.0.1:48741

Common launch patterns

```
hermes serve --host 127.0.0.1 --port 48741
hermes serve --host 0.0.0.0 --port 48741 --auth-token <token>
ssh -N -L 48741:127.0.0.1:48741 user@your-vps
```

Keep public VPS access behind HTTPS, bearer-token authentication, CORS allowlists, workspace sandboxing, upload limits, audit logs, and server-side archive cleanup. SSH tunnel mode is preferred when a user does not need a public daemon URL.

Settings: runtime target

The runtime target controls sit in Settings with the same visual treatment as appearance and container runtime controls.

Where users configure it

- Open Settings, then use the runtime area below Light/Dark appearance and near the Docker Desktop / Colima container options.
- A horizontal divider separates the IDE connection target from the local container provider preferences.
- Current macOS users keep the default local daemon behavior unless they explicitly switch modes.

Control	Meaning
Runtime mode	Choose local desktop, remote VPS, or SSH tunnel. The UI uses the same DaemonClient boundary for all modes.
Daemon URL	The base URL for the selected daemon. Local and SSH tunnel modes usually use 127.0.0.1. Remote VPS mode should use HTTPS.
Bearer token	Optional token attached to daemon requests. Required when the daemon is launched with --auth-token.
Test connection	Calls /health and shows capabilities such as workspace import, files, terminal, preview, and agent run support.
Reset local	Restores the legacy local macOS-safe default without touching container provider preferences.

Daemon health contract

```
GET /health
{
  "ok": true,
  "name": "hermes-runtime",
  "capabilities": ["workspace.importZip", "terminal.run", "agent.run"]
}
```

ZIP import and upload hygiene

Users can drag a ZIP into the chat pane or choose Import code ZIP from the plus menu. The archive is cleaned before it reaches the daemon import path.

Import entry points

Drag and drop a project ZIP onto Agentic Development, or open the plus menu and choose Import code ZIP. The chat is prefilled so the user can immediately ask the agent to inspect or modify the imported workspace.

Safe templates stay

Files such as .env.example, .env.sample, .env.template, and .env.dist remain because they document configuration without carrying live secrets.

Cleaned before upload/import

- .DS_Store files and __MACOSX folders are removed.
- AppleDouble resource fork files such as ._README.md are removed.
- .env, .env.local, .env.production, .envrc, and similar secret-bearing files are stripped.
- Unsafe absolute paths and ../ zip-slip paths are rejected.
- Only the sanitized archive bytes are sent to the daemon. The original local archive path is not sent as an import source.

Daemon-side rule

Client-side cleanup improves UX, but the daemon must still treat every uploaded archive as untrusted. Remote VPS deployments must repeat archive validation server-side, enforce size limits, and keep imported workspaces isolated from the rest of the host.

Client downloads and S3 releases

The /ide page now presents a small download menu instead of a single direct ZIP-style action so users can choose the right client for their environment.

Download menu behavior

- The primary Download BuilderStudio button opens a compact in-button menu with macOS, Linux, and Windows clients.
- The menu is fed by the macOS release manifest and the cross-platform release manifest when available.
- The visible download card still updates when a user selects a client, so platform, architecture, format, size, and checksum remain reviewable.

Artifact	Typical user	Notes
macOS DMG / ZIP	Existing Mac users	Existing signed and notarized macOS flow stays the default path.
Linux AppImage / DEB / RPM / tar.gz	Linux desktop and VPS operators	Good for local Linux workstations and admins managing remote Hermes runtimes.
Windows EXE / MSI / ZIP	Windows developers	Connect locally, to a remote VPS daemon, or through an SSH tunnel.
release-manifest.json	Website and automation	Published to S3 latest and versioned prefixes so the site can discover available clients.

Release flow rule

Cross-platform packaging and S3 publishing are additive. They should not replace or break the current macOS release, signing, notarization, or upload flow. The website can discover new artifacts from the S3 manifest as they become available.

Full-stack, Workflow-aware, Iterative App Building

This update explains the newer build approach: chat can create full-stack products, Workflows can become implemented app modules, and follow-up prompts can continue improving the same generated product instead of restarting from a blank app.

What changed

BuilderStudio now treats app building as a loop: prompt, generate, auto-wire backend when needed, run preview, repair failures, then iterate module by module.

Who this helps

Users who want to build real products from chat, turn business processes into software, and keep adding features after the first generated app exists.

```
New app -> UI + backend intent detection -> Supabase/Postgres scaffold
       -> local preview + backend start -> Doctor/auto-heal if broken
       -> /iterate follow-up modules -> Workflow handoff -> deploy preflight
```

Recommended demo path

- Run /demo and show the full-stack app prompt, Workflows handoff, /iterate follow-up, Doctor repair context, Hermes/Aurelius guide, and deployment preflight.
- Use one sample product throughout the demo so users see that BuilderStudio keeps building the same product instead of creating unrelated one-shot projects.
- Explain that approvals still protect writes, installs, terminal commands, backend starts, and deployment actions.

Full-stack app auto-wire

When a user asks for a full-stack app, BuilderStudio should connect the generated UI to backend and database scaffolding automatically. The goal is to let users describe product behavior in plain language while the workspace receives the files, environment template, local backend commands, and preview wiring needed to run it.

Trigger phrases

Prompts that mention full-stack, Supabase, Postgres, auth, database, CRUD, API, records, dashboards, tickets, invoices, teams, profiles, workflow logs, approvals, or persistent state should activate backend auto-wire.

Generated target

Backend files should be created inside the generated app folder, not a disconnected workspace root, so the UI, schema, client helpers, env example, and runtime preview stay together.

What users should understand

- They do not need to open the Backend Services panel first when the chat request clearly asks for a full-stack app.
- The panel remains useful after generation because it explains running state, local URLs, generated files, env references, type generation, and Supabase Studio access.
- Secrets remain separated from browser-safe values. Frontend code should only receive public anon/client values such as VITE_SUPABASE_URL and VITE_SUPABASE_ANON_KEY.
- If Supabase is not available, a fallback Postgres path can keep local data work moving while setup is completed.

```
Example prompt:
Build a full-stack customer support portal with Supabase auth, Postgres tables for
  accounts, tickets, comments, and invoices, a responsive dashboard, admin
  filters, and a local preview.
```

Workflows as product blueprints

Workflows should be presented as a bridge from process design to real software. A workflow diagram is not only a diagram: it can become a product module with screens, APIs, persistence, run history, approvals, retries, and connector contracts.

Workflow handoff	Use Send to AI builder after designing a trigger, actions, logic gates, payload, expected outcome, and guardrails. BuilderStudio stores the workflow as active build context and fills the Agentic Development chat with an implementation prompt.

Workflow-to-app steps

- Open /workflow or the Workflows pane in Agentic Development.
- Choose a template or use Workflow Director to describe the automation.
- Review the contract: goal, input payload, expected outcome, approvals, secrets, and MCP exposure if needed.
- Click Send to AI builder. The next chat run should implement the workflow in the current/generated app.
- Use /iterate for follow-up modules such as better run logs, approval queues, analytics, connector setup, or notification rules.

```
Example workflow handoff:
Route high-risk orders through approval before fulfillment. Persist runs, show
  approval queue, retry failed webhook steps, and expose an audit trail in the
  admin dashboard.
```

Module-by-module iteration

The new app-building loop should teach users to start with a strong first product, then keep adding scoped modules. This makes BuilderStudio feel less like one big generation and more like a working product studio.

Use /iterate

Type /iterate followed by one focused request. BuilderStudio targets the latest generated app folder, asks the agent to inspect the existing code, and preserves unrelated behavior.

Automatic follow-ups

Plain follow-up prompts that look like additive product work can also run in iteration mode when a latest generated app exists.

Good iteration prompts

- /iterate add a billing settings module with plans, invoices, and usage limits.
- /iterate add audit logs for user, workflow, and billing events.
- /iterate connect the approval workflow to the existing dashboard navigation.
- /iterate add empty/loading/error states for the tickets table without redesigning the landing page.
- /iterate add tests or validation commands for the auth and tickets modules.

Best practice

Keep each iteration scoped to one coherent module. The agent should report the module boundary, changed files, commands run, validation result, and any follow-up risks. Avoid broad visual rewrites unless the user explicitly asks for a redesign.

Runtime Doctor and repair loop

When generated apps break during install, compile, run, or preview, the repair loop should make failure evidence useful to the next agent pass. The user should see that BuilderStudio is not just retrying blindly.

Doctor evidence

Collect compiler output, missing module errors, runtime start failures, preview notes, dependency drift, and known Vite/import-analysis failures.

Prompt memory

Create a repair appendix and append it to the next chat prompt so the model receives the original request plus the concrete failure that must be fixed.

User-facing explanation

- Doctor tries safe auto-heal first for common dependency, import, package manager, and runtime-start issues.
- If the preview is still broken, the repair loop saves the failure context and prepares a better next prompt.
- The next agent pass should fix the specific compiler or runtime issue, rerun targeted validation, and keep unrelated modules stable.
- Future visual repair can add broken-screen screenshot evidence when the preview frame can be captured safely.

```
Repair prompt shape:
Original request + generated app path + compiler/runtime evidence
+ broken preview notes + Doctor diagnosis + required fix checklist
```

Interactive demo and guide updates

The demo should explain the same product-building loop documented in this guide, including explicit runtime boundaries and optional Fusion.

/demo walkthrough

Show workspace import, machine setup, Pathways, Conversation and Plan modes, a full-stack prompt, workspace-boundary checks, workflow handoff, /iterate, simulator persistence, Fusion configuration, explicit /hermes use, validation, and deployment preflight.

Slash commands to teach

Command	Behavior
/demo	Replay the guided walkthrough with the current build and iteration loop.
/conversation	Discuss ideas and requirements without edits or execution.
/plan	Produce an implementation plan and validation checklist.
/build	Return to normal approval-aware daemon execution.
/workflow	Open visual workflows and hand a process to the builder.
/iterate	Patch the latest generated app inside the selected workspace.
/fusion	Open or control opt-in OpenRouter Fusion.
/hermes	Open or run the contained runner explicitly.
/aurelius	Open or run planned Aurelius orchestration.

Example demo script

Use this script when onboarding users or recording a product demo. Keep the same sample product throughout so the audience sees continuous product evolution.

1. Start with intent	Ask for a full-stack customer portal with auth, tickets, invoices, and a dashboard.
2. Show auto-wire	Point out that backend files, env examples, database schema, and preview startup are scoped to the generated app folder.
3. Open Workflows	Create an approval workflow and send it to the AI builder as a product module.
4. Iterate	Use /iterate add billing settings, then explain that the agent patches the existing product instead of rebuilding.
5. Break/fix story	Explain that Doctor captures compiler/runtime evidence and feeds a repair appendix into the next prompt when a run breaks.
6. Deploy readiness	Run deployment preflight only after UI, backend notes, and runtime validation look healthy.

```
One-line product story:
BuilderStudio helps users build real apps from chat, wire backend data
  automatically, turn workflows into modules, and keep improving the same product
  through safe iterative loops.
```

Conversation, Plan, and Build Modes

These modes make build-free intent explicit and reduce accidental writes or runtime actions.

/conversation - discuss without building

Use for discovery, requirements, alternatives, debugging discussion, UX review, or tradeoffs. No file writes, terminal commands, runtimes, packaging, or deployment.

/plan - plan before acting

Use for milestones, file-level strategy, risks, dependencies, acceptance criteria, and validation steps before any build begins.

/build - allow normal builder behavior

Use when the user is ready for workspace context, capability requests, edits, terminal checks, previews, packaging, or deployment handoff.

Mode behavior

```
User intent -> /conversation or /plan -> answer only
User intent -> /build -> approvals -> BuilderStudio daemon execution
            -> changed files, diagnostics, simulator preview, package, or handoff
User intent -> /hermes -> explicit contained Hermes execution
```

- Normal visual generation remains on the simulator/runtime canvas.
- Output is selected automatically only for backend-only work without a visual preview.

Project targeting, /iterate, and /fresh

BuilderStudio now distinguishes editing the current project from intentionally starting a new app. The active terminal directory, latest generated app path, and project detection signals guide where files are written.

Current project iteration

/iterate patches the current project directory when the active terminal is already inside a detected app. It should not create nested app folders for normal follow-up work.

Fresh project sessions

/fresh starts a new app/session. If the user is inside an existing project, BuilderStudio moves one level up and creates the new app beside the current project instead of inside it.

Blank directories

When the selected/current directory is genuinely blank, BuilderStudio can write the first generated app directly into that directory.

Project containers

When a folder contains one obvious child project, /iterate targets that child. When it contains several projects, BuilderStudio should ask or require an explicit target.

Decision table

Current directory	Command	Expected target
Active project	/iterate	Patch the current project in place
Active project	/fresh	Use the parent folder and create a sibling app
Blank directory	First build or /fresh	Write the app directly into the blank directory
Project container	/iterate	Target the detected child project or ask if ambiguous
Any target	Writes	Run daemon write preflight before applying files

Examples

```
/iterate change the developer checkpoint from npm commands to Yarn commands
  -> patches the current detected project

/fresh create a CRM dashboard
  -> if currently inside my-old-app/, creates a new sibling beside my-old-app/
```

Workspace safety and write reliability

The selected workspace is the hard trust boundary. BuilderStudio never assumes a particular folder name such as /dev and never promotes a workspace to its parent.

Selected workspace boundary

Relative targets resolve under the selected root. Absolute targets outside it, parent traversal, same-prefix siblings, and stale generated-app roots are rejected.

Defense in depth

The frontend stops unsafe generation before writing. The daemon independently validates the canonical target against the trusted workspace.

Write preflight

Before applying files, the daemon can create, read, and delete a probe under .builderstudio in the selected target to prove the directory is writable.

Stale preview protection

A failed write or validation run must not look like a successful new build. Keep the prior preview labeled as stale until a verified run completes.

Path examples

```
POSIX workspace:  /home/username/projects/sample-suite
Allowed target:   /home/username/projects/sample-suite/sample-web
Rejected sibling: /home/username/projects/sample-suite-old
Windows workspace: C:\Users\username\projects\sample-suite
Rejected target:   C:\Users\username\projects\other-app
```

Updated command guidance

Use these commands to teach the difference between discussion, planning, editing the current app, starting fresh, and running contained multi-agent work.

Command	Use it for
/conversation	Discuss requirements, tradeoffs, UX, or debugging without writing files.
/plan	Create an implementation plan, risks, dependencies, and validation checklist without building.
/build	Return to normal approval-aware editing, commands, runtimes, packages, and previews.
/iterate	Patch the current detected project or latest generated app instead of creating a new folder.
/fresh	Start a new app/session. From inside a project, create the new app in the parent folder.
/swarm	Run contained parallel lanes from the current host workspace mounted as /workspace.

Recommended teaching script

- Open or create a trusted workspace. BuilderStudio inspects project signals before deciding where files belong.
- Use /iterate when the user wants to keep improving the same product. The current project should be patched in place.
- Use /fresh when the user wants a separate new app while standing inside an existing project.
- Use /swarm when the goal benefits from independent lanes, each mounted from the current host workspace into a contained /workspace path.
- After any failed write, explain that the preview may still be the previous successful run until a new build token is verified.

Quick prompt examples

```
/iterate add loading and empty states to the existing dashboard
/iterate replace npm install / npm run dev with Yarn commands in the UI copy
/fresh create a new analytics dashboard for marketplace sellers
/swarm run a QA, accessibility, security, and implementation review of this project
```

Project navigation with /list and /cd

BuilderStudio can help users move between local projects without leaving the Agentic Development flow. These commands use the same project detection and trusted workspace rules as /iterate and /fresh.

/list

Lists immediate child directories that look like projects. If the terminal is already inside an active project, BuilderStudio lists sibling projects from the parent folder instead of showing the current project internals.

/cd <project>

Switches the active workspace and terminal current directory to a listed project. The command supports exact names and unique prefixes, and rejects ambiguous matches with a clear list of options.

Base directory behavior

When the current directory is a project, the base directory for navigation becomes its parent. This makes it easy to jump between sibling apps created with /fresh or opened from the same development folder.

Safety boundary

Project navigation still goes through the daemon and trusted workspace checks. BuilderStudio should not switch into arbitrary system paths or write outside the user-approved local workspace area.

```
Current terminal cwd:
  /Users/username/development/sample-suite/sample-web

User runs:
  /list

BuilderStudio lists sibling projects from:
  /Users/username/development/sample-suite

Example results:
  sample-web/
  sample-api/
  sample-admin/
```

Autocomplete and project switching examples

Use Tab completion with /cd to reduce typing and avoid selecting the wrong project. BuilderStudio should complete only when there is a safe, unique directory match.

Tab completion

Type /cd followed by the beginning of a project directory name, then press Tab. If there is exactly one match, BuilderStudio fills the rest of the directory name.

Ambiguous matches

If several projects share the same prefix, BuilderStudio should not guess. It should print the matching project names and wait for the user to type more characters.

Exact and prefix matching

Typing /cd crm-dashboard switches directly to that project. Typing /cd crm can also switch there when crm-dashboard is the only project beginning with crm.

How this fits with /fresh

/fresh creates a new project beside the current one. /list shows those sibling projects. /cd moves between them. /iterate then edits the currently selected project.

Common examples

```
/list
  Shows projects in the current project container or parent folder.

/cd crm<Tab>
  Completes to /cd crm-dashboard when it is the only match.

/cd sigidb-admin
  Opens sigidb-admin as the active workspace and terminal cwd.

/fresh create an analytics console
  Starts a new sibling project from inside an existing project.
```

Command System Update

New documentation for /repair, /root, project navigation, runtime repair loops, and the reconciled slash-command set.

Why this update exists

BuilderStudio already documents the core loop: open a trusted workspace, choose a model, attach skills/pathways/MCP/connectors, ask the agent to plan or build, approve risky actions, and review changed files, diagnostics, commands, previews, and output. This addendum updates the guide for the newer recovery and navigation flows that make that loop easier to operate.

Updated command model

Mode commands

/conversation, /plan, and /build choose whether the AI talks, plans, or starts the approval-aware build loop.

Action commands

/repair, /workflow, /terminal, /git, /deploy, /demo, /setup, and similar commands open panels or run bounded flows.

Navigation commands

/pwd, /root, /ls, /list, and /cd keep users from getting trapped in nested generated-app folders.

Recommended recovery path

```
/pwd
/root
/list
/cd sample-app
/repair
```

Safety principle

/repair and related runtime fixes should prefer the smallest deterministic source change that makes the current app compile and render. Do not rewrite unrelated UI unless the diagnostics require it.

Core user-facing commands

These are the commands most users should see in the chat slash-command menu and command palette.

Command	Use it for	Behavior
`/conversation`	**Conversation mode**	Chat, brainstorm, clarify, debug, or review tradeoffs without starting a build.
`/plan`	**Plan mode**	Create an implementation plan, milestones, risks, dependencies, and validation checklist without file edits.
`/build`	**Build mode**	Return to normal approval-aware build behavior for edits, terminal checks, previews, packaging, or deploy handoff.
`/fresh <request>`	**Fresh app**	Create a new app beside the current project instead of nesting it inside the active project.
`/iterate <request>`	**Iterate current app**	Patch the current/latest generated project in place with the smallest coherent module or fix.
`/repair [--dir path] [notes]`	**Doctor repair**	Run deterministic repair against the active/generated app, then restart preview and queue model repair context if needed.
`/pwd`	**Current directory**	Show selected workspace, active terminal directory, latest generated app memory, and current BuilderStudio directory.
`/root`	**Return to root**	Return to the remembered project-start root/base folder after working inside nested generated projects.
`/ls [path]`	**List files**	List files and folders in the current directory or a requested child path.
`/list [path]`	**List projects**	List nearby/sibling project directories using the same project detection rules as /iterate and /fresh.
`/cd <project>`	**Switch project**	Switch the selected workspace/current directory to a listed project. Terminal panes can use Tab completion.
`/workflow`	**Open workflows**	Open the visual workflow designer and use it as implementation context for app modules.
`/simulator`	**Open simulator**	Switch the right pane to the local runtime preview. Alias: /preview.
`/terminal`	**Open terminal**	Open terminal/output panes. Alias: /output.

Core user-facing commands, continued

Surface, Git, runtime, session, demo, and setup commands.

Command	Use it for	Behavior
`/git`	**Open Git**	Open repo import, GitHub connection, status, diff, commit, and push controls. Alias: /github.
`/diff`	**Analyze diffs**	Open Git diff/conflict analysis. Alias: /conflicts.
`/autoresolve`	**Resolve conflicts**	Prepare an LLM-assisted merge-conflict resolution prompt from the current diff. Alias: /resolve.
`/push`	**Quick push**	After approval, run the quick push flow for the current repository.
`/deploy`	**Deployment**	Open deployment readiness and handoff for the selected workspace. Alias: /shipyou.
`/colima`	**Use Colima**	Prefer Colima-backed container execution for generated-app install/build/preview.
`/docker`	**Setup Docker**	Open Docker Desktop setup, start/check the daemon, and prepare container-backed builds.
`/reset`	**Reset context**	Reset chat/token context, pending approvals, runtime state, and selected run state while preserving workspace.
`/clear`	**Clear chat**	Start a fresh chat/token context while keeping the selected workspace open.
`/demo`	**Guided demo**	Run the guided product tour.
`/start`	**Getting started**	Replay first-run onboarding. Alias: /getting-started.
`/pathways`	**Pathways**	Browse starter prompt cards and matching skills.
`/setup`	**Machine setup**	Prepare local developer toolchain dependencies.
`/helloworld`	**Hello World**	Load the smallest first-app walkthrough.

/repair and Doctor repair flow

Use this when an app fails to compile, starts with a white screen, has missing Vite/React entrypoints, or fails runtime start.

```
/repair
/repair white screen after runtime start
/repair --dir /path/to/generated-app failed Vite preview
```

What Doctor can patch

Root index.html, src/main.jsx or src/main.tsx, package scripts, Vite React dependencies, static server behavior, missing build/preview scripts, package-manager selection, and lockfile conflicts.

What Doctor records

Compiler logs, runtime logs, terminal output, preview evidence, target directory, runtime URL, and focused repair notes in .builderstudio/runtime-repair-loop.md.

Repair sequence

1. Resolve the active/generated app directory, or use --dir if the user provides one.
2. Run deterministic checks before asking the model to guess.
3. Reconcile package manager and lock files before install/start validation.
4. Apply safe source patches and restart the preview.
5. If still broken, queue a focused model repair prompt using the captured failure evidence.

Package manager selection

```
pnpm-lock.yaml      -> pnpm install
package-lock.json   -> npm install --no-audit --no-fund
npm-shrinkwrap.json -> npm install --no-audit --no-fund
yarn.lock           -> yarn install --non-interactive
bun.lock/bun.lockb  -> bun install when available, otherwise npm
no lockfile         -> packageManager field if declared, otherwise npm
```

/root and project navigation

Use these commands to move safely between the base workspace, generated apps, sibling projects, and selected runtime targets.

Common problem

Users often generate one or more apps under a development directory, then continue working while BuilderStudio is pointed at a nested generated-app folder. /root, /list, and /cd make the current target explicit before running /iterate, /fresh, /repair, or runtime start.

Command	Use it for	Behavior
`/pwd`	**Show active target**	Display the selected workspace, active terminal directory, latest generated app, and current BuilderStudio directory.
`/root`	**Return to base**	Reopen the remembered project-start root/base folder, such as the parent dev directory.
`/ls [path]`	**Inspect files**	List files/folders in the current directory or a requested child path.
`/list [path]`	**Find projects**	List nearby project folders, usually siblings of the current generated app.
`/cd <project>`	**Switch target**	Switch the selected workspace/current directory to one listed project. Terminal panes can use Tab completion.

Example navigation session

```
/pwd
/root
/list
/cd crm-dashboard
/iterate add loading, empty, and error states
/repair
```

Build continuation flows

Use /fresh when starting a separate new product. Use /iterate when continuing the current product.

/fresh

Starts a new app/session beside the current project. Use it when the user wants a separate product and should not mutate the app currently open.

/iterate

Patches the current/latest generated app in place. Use it for focused modules, improvements, tests, UI states, bug fixes, and follow-up product work.

```
/fresh create a CRM dashboard for boutique agencies
/iterate add billing settings with plans, invoices, usage limits, and audit events
/iterate fix empty and loading states without redesigning the dashboard
```

Workflow-to-builder handoff

The /workflow command opens the visual workflow designer. When the user sends a workflow to the AI builder, BuilderStudio should treat the workflow as implementation context for the current/generated product: UI, API/data state, logs, approvals, and follow-up iteration should remain connected.

Targeting rule

Before file writes, BuilderStudio should confirm whether the target is the root workspace, a sibling project, the latest generated app, or a user-provided --dir path. Use /pwd and /list when the target is unclear.

Agent runtime commands

BuilderStudio daemon execution is the normal default. Aurelius and Hermes are explicit command paths with separate contracts.

Command	Use it for	Behavior
/aurelius	Agent runtime	Open the Aurelius command center.
/aurelius help	Agent runtime	Show Aurelius commands and boundaries.
/aurelius status	Agent runtime	Show the built-in runtime and container contract.
/aurelius run <request>	Agent runtime	Run Aurelius against the selected workspace.
/aurelius swarm <goal>	Agent runtime	Run an Aurelius-led multi-lane flow without implicit Hermes.
/aurelius context	Agent runtime	Explain orchestration, daemon enforcement, and approvals.
/hermes	Agent runtime	Open Hermes guide and status. This does not start a run.
/hermes help	Agent runtime	Show the explicit command order and contained-runner workflow.
/hermes status	Agent runtime	Check container runtime, image, provider config, and boundary.

Agent runtime and Fusion commands, continued

Hermes remains command-only, swarms do not fall back to Hermes, and Fusion is an independent OpenRouter option.

Command	Use it for	Behavior
/hermes install	Agent runtime	Prepare the local Hermes runner image.
/hermes setup	Agent runtime	Save non-interactive provider/model configuration.
/hermes run <request>	Agent runtime	Launch Hermes with the required explicit invocation marker.
/hermes context	Agent runtime	Explain containment, workspace mount, context, and reset behavior.
/swarm	Agent runtime	Open or start a multi-lane flow using the selected supported runtime.
/fusion	OpenRouter	Open Fusion configuration. Alias: /fusion configure.
/fusion on	OpenRouter	Enable Fusion with the saved mode and model panel.
/fusion auto	OpenRouter	Let the outer model decide when to deliberate.
/fusion always	OpenRouter	Require the Fusion tool on each eligible request.
/fusion status	OpenRouter	Show enabled state, mode, analysis models, judge, and web limit.
/fusion off	OpenRouter	Disable Fusion without changing the selected outer model.

Default agentic orchestration

Nontrivial code requests can use guided discovery, planning, execution, and validation without changing the runtime to Hermes.

1. Discover

Inspect the selected workspace, project manifests, scripts, frameworks, and repository boundaries before choosing a write target.

2. Resolve the target

Explicit project names and the selected workspace take priority. Ambiguous roots show project choices instead of continuing stale work.

3. Create a contract

Record the operation, canonical target, permitted write roots, protected paths, package manager, validation plan, preview needs, risks, and assumptions.

4. Implement and verify

Use BuilderStudio daemon execution or the explicitly selected supported runtime, then run project-aware validation and a focused repair pass only when evidence requires it.

What remains explicit

- Conversation and Plan modes remain build-free. Use /build when edits or execution are intended.
- Hermes is excluded unless the request comes from /hermes.
- Writes, installs, commands, runtimes, external tools, and deployment remain subject to approvals and workspace trust.

```
workspace discovery -> target resolution -> execution contract
-> approval gates -> daemon implementation -> validation
-> focused repair when needed -> verified completion report
```

How project targeting works

BuilderStudio chooses a target before writing and treats the selected workspace root as a non-negotiable boundary.

Explicit project name

When a request names an existing project under the selected workspace, that project wins. If the project cannot be found, pause for target selection.

Selected project workspace

When the selected workspace is already a project, build and iterate in place unless the user explicitly asks for a separate app inside that workspace.

Workspace with many projects

Score candidate directories from manifests, scripts, framework hints, recent context, and explicit names. Ask when candidates are ambiguous.

No parent promotion or stale continuation

/fresh does not move the workspace to its parent. A previous generated-app path outside the selected root is ignored and never silently resumed.

Target priority

```
1. Explicit absolute path inside the selected workspace
2. Explicit named project inside the selected workspace
3. Currently selected project workspace
4. Explicit /fresh target inside the selected workspace
5. Best-ranked candidate from an unambiguous project map
6. Ask the user when more than one candidate remains
```

Execution contracts and opt-in automation

The agent may inspect broadly, but it writes and runs commands only inside an approved execution contract.

Mutation envelope

The contract defines exact paths, allowed directory prefixes, permitted file extensions, protected paths, maximum change size, and whether package or lock files may change.

Project-aware validation

Validation comes from the actual repository: workspace scripts, type checks, tests, builds, API health checks, or library checks. A browser preview is not assumed for backend-only work.

Automatic behavior is off by default

New app scaffolding, database provisioning, dependency installation, Docker runtime, browser preview, backend auto-wire, and deployment must be justified by the contract.

Scope expansion requires review

If implementation needs additional directories, dependencies, or external services, BuilderStudio should explain why and request approval rather than silently widening access.

Files written for agentic bookkeeping

```
.builderstudio/agent-workspace-discovery.json
.builderstudio/agent-orchestration.json
.builderstudio/agent-execution-contract.seed.json
.builderstudio/agent-execution-contract.json
.builderstudio/agent-validation-report.json
```

Important rule

- No preview, database, install, or runtime should start merely because a prompt contains words such as API, database, dashboard, or full-stack.
- The execution contract must opt into those capabilities using workspace evidence and the user request.

Validation, recovery, and user guidance

BuilderStudio should surface evidence and safe next actions instead of silently retrying unrelated workflows or changing canvases without a reason.

Failure or state	Guided behavior
Target problem	Return to target discovery or present project choices.
Permission needed	Show the exact action, risk, working directory, and approval controls.
Mutation violation	Stop and show which canonical path exceeded the approved envelope.
Typecheck or test failure	Run one focused repair pass using the actual diagnostics.
Visual generation	Keep the simulator/runtime canvas selected throughout generation and validation.
Backend-only request	Validate backend scripts or API startup; output may be selected because no visual preview applies.
No runtime final message	Recover changed files, check policy, validate, and synthesize the report.

Completion states

- Verified - required validation passed and the completion notification may be shown.
- Unverified - files changed, but required validation was unavailable or did not run; say so clearly.
- Needs repair - validation produced actionable diagnostics and a bounded repair pass is available.
- Stopped safely - the target, permission, runtime, or mutation boundary was not acceptable.

Agentic controls and command chaining

Commands can be combined on one line, and the agentic orchestration preference can be inspected or changed when needed.

Command	Behavior
`/agentic`	Show the current default-agentic setting and usage.
`/agentic status`	Report whether default agentic orchestration is enabled.
`/agentic on`	Enable discovery, contracts, and project-aware validation by default.
`/agentic off`	Use the legacy build path for troubleshooting or controlled compatibility.
`/pet`	Show the global resting desktop pet and enable completion celebrations.
`/petstop`	Hide the pet and disable pet completion announcements.

Command chaining

- Recognized slash commands can be placed on one line and are executed sequentially, not concurrently.
- Text following a command remains attached to the command that accepts a request.
- Approvals still pause the sequence when a command requires user confirmation.

```
/agentic status /pwd
/pet /fresh create a simple inventory dashboard
/root /list /cd sample-api
/clear /plan review the authentication architecture
```

Use with care

Do not chain commands when the first command must produce information that changes what the next command should do. Run those steps separately so the target remains visible.

Desktop pet, notifications, and approvals

Build completion and approval requests can guide users back to BuilderStudio without relying on the chat window alone.

Resting and celebrating pet

Running /pet shows a still resting Bongo Cat. The animation plays only after a verified build completes, then returns to rest when the user focuses the app or starts typing.

System notifications

When permitted by the operating system, BuilderStudio can show native notifications for completed builds and approval-required events. Selecting a notification returns to Agentic Development.

In-app alerts

A visible alert provides a reliable fallback when native notifications are unavailable. Approval alerts remain until handled or dismissed; completion alerts may expire automatically.

App badge and attention

Background events can increment the app badge or taskbar overlay. Returning to BuilderStudio clears the unread badge while retaining useful in-app alert history.

Recommended notification behavior

- Build completion is normal priority and should only fire after required validation passes.
- Approval requests are higher priority because the agent cannot continue without the user.
- Duplicate alerts should be suppressed so one pending approval does not produce repeated notifications.
- Notification text must not include secrets, plaintext sensitive data, API keys, full personal paths, or private prompt contents.

```
Typical approval notification:
BuilderStudio needs approval to run a project command.
Open Agentic Development to review the action and working directory.
```

Privacy-safe documentation and support sharin

Documentation, screenshots, logs, and support bundles should use neutral examples and remove personal or secret information before they leave the device.

Use placeholders

Use names such as username, sample-project, example.com, demo-user, and demo-tenant. Avoid real names, email addresses, organization identifiers, home-directory names, or customer data.

Sanitize local paths

Replace personal paths with forms such as /Users/username/development/sample-project or /home/username/projects/sample-project before publishing screenshots or command output.

Remove secrets and identifiers

Never include API keys, access tokens, cookies, OAuth codes, private repository URLs, signing identities, machine names, IP addresses that identify private infrastructure, or unredacted environment values.

Keep useful diagnostics

Preserve error categories, command names, exit codes, package versions, generic relative paths, and concise stack traces. These usually provide enough evidence without exposing identity.

Before sharing a support bundle

- Search case-insensitively for personal names, usernames, email addresses, home-directory paths, repository owners, and company-internal project names.
- Replace absolute paths with a neutral placeholder while keeping the relevant relative path.
- Review screenshots as well as searchable text and document metadata.
- Confirm that redaction removes the underlying text instead of drawing a visible box over searchable content.

```
Avoid: /Users/real-name/client-work/private-api
Use:   /Users/username/development/sample-api

Avoid: real-person@example.com
Use:   developer@example.com
```

Workspace boundary enforcement

The selected workspace root is the security and write boundary for every user, on POSIX and Windows systems. No folder name is special.

Canonical containment

Resolve the selected workspace and requested target to canonical paths before writing. The target must equal the root or be a true descendant with a path-separator boundary.

Rejected target forms

Reject parent traversal, absolute outside paths, same-prefix siblings, symlink escapes, saved generated roots outside the workspace, and implicit parent promotion.

Frontend guard

Stop generation before any file application when the target is outside the selected workspace. Show the selected root and rejected target in neutral, privacy-safe form.

Daemon guard

Repeat the same containment test in the daemon. Client validation is convenience; daemon validation is the authority for filesystem access.

Containment examples

```
root:    /home/username/work/sample-suite
allow:   /home/username/work/sample-suite/apps/web
reject:  /home/username/work/sample-suite-old
reject:  /home/username/work/other-app
root:    C:\Users\username\work\sample-suite
allow:   C:\Users\username\work\sample-suite\apps\web
reject:  C:\Users\username\work\other-app
```

Simulator and output behavior

Canvas selection should reflect the work being performed, not merely the fact that generation is in progress.

Visual builds stay on the simulator

When a build creates or changes a visual application, keep the runtime/simulator canvas selected while files are generated, commands run, validation completes, and the preview refreshes.

When Output is appropriate

Select Output automatically only when the operation is backend-only, produces no usable visual preview, or the user explicitly chooses Output.

Avoid false transitions

Do not switch canvases because a transcript, diagnostic, or completion report exists. Those results remain accessible without replacing the active simulator.

Expected sequence

```
visual request -> simulator selected -> generation -> validation -> preview refresh
backend-only request -> output selected -> command/API validation -> completion report
user canvas selection -> preserved unless the requested operation requires another surface
```

- A stale preview must remain labeled stale after a failed write or failed validation.
- A verified preview may replace the stale state only after the latest build token and runtime URL match.

OpenRouter Fusion workflow

Fusion is an opt-in multi-model deliberation layer for OpenRouter requests. It does not change workspace permissions or enable another agent runtime.

Outer model stays selected

BuilderStudio keeps the exact OpenRouter model chosen by the user as the outer response model and attaches the Fusion server tool to eligible requests.

Analysis panel and judge

Choose 1-8 analysis models. Optionally choose a judge model, or use the selected outer model as the judge when that behavior is configured.

Auto or Always

Auto lets the outer model decide whether Fusion is needed. Always requires the Fusion tool for each eligible request.

Controlled web access

Set a bounded web search/fetch allowance from 1-16 calls. This is separate from local file, terminal, connector, or deployment permissions.

Workflow coverage

- Conversations and planning requests when Fusion is enabled.
- Normal generation, existing-app iteration, transformations, and repair calls.
- Workflow handoffs that use the selected OpenRouter profile.
- No automatic Hermes, Aurelius, orchestration, provider fallback, or expanded workspace access.

Fusion configuration, cost, and privacy

The Fusion modal and slash commands make multi-model deliberation visible, reversible, and bounded.

Command	Behavior
/fusion or /fusion configure	Open the configuration modal.
/fusion on	Enable Fusion with saved settings.
/fusion auto	Enable Auto deliberation mode.
/fusion always	Require Fusion on each eligible request.
/fusion status	Show mode, model panel, judge, web limit, and enabled state.
/fusion off	Disable Fusion while preserving the selected outer model.

Cost and privacy guidance

- Fusion can add analysis-model, judge-model, and web-tool usage in addition to the outer-model request.
- Show the enabled state and selected models before a run so the user can understand the expected scope and cost.
- Do not place API keys, personal names, email addresses, full private paths, or unredacted prompt contents in screenshots, logs, notifications, or documentation examples.
- Use neutral placeholders such as username, sample-project, developer@example.com, and example.com.

Trackless, Contract-First Builds

This update documents the new build model: BuilderStudio no longer treats a selected track as a structure lock. The prompt, repository context, and generated project contract determine how a project is created, validated, previewed, and run.

What changed

Legacy build tracks are compatibility hints only. They may bias generation, but they do not force Vite, React, Yarn, Docker, browser preview, or localhost rendering.

The generated project is expected to declare its own runtime truth in .builderstudio/project-contract.json.

The daemon starts from the contract first, then falls back to legacy/native or file detection only when a contract is absent.

```
Old flow:
selected track -> force scaffold/runtime -> generated files -> assumed preview

New flow:
prompt + repo context -> model chooses project shape -> project contract -> runtime
  adapter -> validation, preview, checkpoint
```

Use this when

The user asks for a native desktop app, mobile app, backend service, CLI tool, library, extension, game, workflow, or other non-web project.

The selected workspace already contains signals that should decide the structure, such as Swift Package.swift, Cargo.toml, pubspec.yaml, pyproject.toml, package.json, or service manifests.

Do not assume

Do not assume a browser preview just because the user says app or dashboard.

Do not install dependencies, start Docker, or run a server unless the contract or approval flow opts into it.

Do not create nested generated app folders when the user is iterating on an existing active project.

Project Contract

Every new generated project should include .builderstudio/project-contract.json. This file is the hard runtime truth for build, validation, preview, and launch.

```
{
  "schemaVersion": 1,
  "projectKind": "browser-web | backend-service | full-stack-app | native-desktop | native-mobile |
    native-cross-platform | cli-tool | library | data-ml | infrastructure | game | extension | custom",
  "runtime": "vite-react | nextjs | fastapi-python | swiftui-macos | tauri | flutter | electron | go-cli |
    rust-cli | custom",
  "previewMode": "browser-url | server-url | desktop-window | device-emulator | hardware-device |
    terminal-process | headless-only | none",
  "hostOnly": false,
  "workingDirectory": ".",
  "install": [],
  "validate": [],
  "build": [],
  "run": [],
  "requiredTools": [],
  "outputs": { "browserUrl": null },
  "notes": ""
}
```

Field	Meaning
projectKind	The broad shape of the project: web, backend, native desktop, CLI, library, game, extension, or custom.
runtime	The specific runtime or framework selected by the model or existing repo context.
previewMode	The UI surface BuilderStudio should expect: browser URL, server URL, desktop window, emulator, terminal, headless, or none.
hostOnly	True when the app must run on the host OS, such as native macOS windows or hardware/device workflows.
install / validate / build / run	Ordered command arrays. The daemon asks for approval where required, then executes inside the approved workspace boundary.
outputs.browserUrl	Set only when a browser or server URL is the correct preview. Native apps should usually leave this null.

Backward compatibility

BuilderStudio should still recognize .builderstudio/native-run.json for older native builds, but project-contract.json wins when both files exist.

Legacy runtime routes may remain for compatibility; the new route is /project/start.

Runtime and Preview Behavior

The runtime canvas should follow previewMode instead of forcing a browser iframe. Browser preview is only one of several possible outputs.

previewMode	Expected BuilderStudio behavior
browser-url	Start the declared web runtime and render the browser URL.
server-url	Start the service and expose/open the declared health, docs, or server URL.
desktop-window	Run host-side and report the native process/window status instead of rendering localhost.
device-emulator	Launch the declared simulator or emulator flow when available and approved.
hardware-device	Run only with explicit approval because external devices may be involved.
terminal-process	Run and show terminal/process output. No visual preview is assumed.
headless-only	Run validation/tests/checks only. No preview surface is expected.
none	Do not start a runtime automatically. Show generated files and instructions.

Host-side native apps

For macOS, Windows, Linux, mobile simulator, and desktop GUI workflows, prefer host-side execution. Docker should not be the default for GUI apps or OS-integrated tooling.

Web and backend apps

For browser and server apps, Docker Desktop, Colima, host Node, or another provider can still be used when the project contract and user approval support it.

```
Runtime route preference:
/project/start        contract-first start path
/runtime/start        compatibility path; should check project-contract.json first
/native/start         compatibility path for older native-run.json projects
```

Native and Non-Browser Generation

Native generation is now a normal project shape, not an exception. The model should choose the right structure from the prompt and repository context, then declare it in the project contract.

```
Example native macOS contract:
{
  "schemaVersion": 1,
  "projectKind": "native-desktop",
  "runtime": "swiftui-macos",
  "previewMode": "desktop-window",
  "hostOnly": true,
  "workingDirectory": ".",
  "validate": ["swift build"],
  "build": ["swift build -c release"],
  "run": ["swift run MacVideoPlayerApp"],
  "requiredTools": ["swift"],
  "outputs": { "browserUrl": null }
}
```

Prompt or repo signal	Likely project shape
native macOS app, SwiftUI, AppKit, menu bar, floating window	native-desktop / swiftui-macos or appkit-macos
Tauri, Electron, Flutter desktop, cross-platform desktop	native-cross-platform / selected runtime
iOS, Android, React Native, Expo, Flutter mobile	native-mobile or react-native-expo-app compatibility
REST API, FastAPI, Express API, worker, service	backend-service
CLI, command tool, package, SDK, library	cli-tool or library
Chrome extension, VS Code extension, game, data pipeline	extension, game, data-ml, or custom

Generation guardrail

Do not add Vite, React, package.json, index.html, browser CSS, localhost preview, or Docker just to satisfy BuilderStudio. Add those only when the chosen project structure actually needs them.

Checkpoint and Continuation

Large apps should be built as durable milestones, not one oversized model response. BuilderStudio can mark a checkpoint, then continue from disk in a later request or a new chat context.

Command	Purpose
/checkpoint optional note	Save durable project state for the active or current generated project.
/continue next thing to build	Read the saved session and continue iterating in the same folder.
/resume next thing to build	Alias for /continue.

```
Checkpoint files:
.builderstudio/build-session.json
.builderstudio/build-session.md
.builderstudio/checkpoints/<timestamp>/summary.md
.builderstudio/checkpoints/<timestamp>/file-list.txt
.builderstudio/checkpoints/<timestamp>/git-status.txt
.builderstudio/checkpoints/<timestamp>/git-diff-stat.txt
```

Automatic checkpoints

After model-authored project files are written, BuilderStudio should save a checkpoint automatically. This protects against chat resets and model-output limits.

Manual checkpoints

Use /checkpoint after a coherent milestone: base architecture, auth, billing, native packaging, tests, docs, or deployment prep.

Milestone Workflow for Massive Apps

Use short, bounded milestones. Each milestone should leave the repo runnable or at least validated, update the project contract when needed, and record a continuation prompt.

```
Recommended flow:
/build create the base project architecture and first runnable screen
/checkpoint base architecture done
/continue add authentication and settings
/checkpoint auth and settings done
/continue add billing and admin dashboard
/checkpoint billing and admin done
/continue polish, tests, docs, packaging, and deployment handoff
```

Milestone rule	Why it matters
One coherent feature set per run	Prevents a single output from exceeding model or writer limits.
Keep project-contract.json current	Lets runtime selection survive chat resets and workspace switches.
Validate before checkpointing when possible	Turns checkpoints into useful restore points instead of snapshots of broken state.
Use continuationPrompt in build-session.json	Gives the next run an exact, bounded next step.
Prefer disk state over chat memory	Allows long projects to continue even when conversation context is cleared.

Good continuation request

/continue implement the next milestone from .builderstudio/build-session.md. Keep the existing runtime and project contract unless the code requires a contract update.

Troubleshooting Contract-First Builds

Use this checklist when generation, writing, or runtime selection does not match the requested project shape.

Symptom	What to check
Model output appears in chat but files are not written	Confirm the managed writer accepted fenced file blocks and that .builderstudio/project-contract.json was included as an actual file block, not just prose.
Native app starts as a browser preview	Check previewMode. Native desktop apps should use desktop-window and browserUrl should usually be null.
Docker tries to run a GUI app	Check hostOnly. Native GUI, simulator, hardware, and OS-integrated workflows should normally set hostOnly true.
Generated app becomes Vite/React unexpectedly	Search for stale Vite hardening guidance or frontend library mode. Tracks should be hints only.
Continuation starts in the wrong folder	Open the intended project workspace or use /workspace open before /continue. Check build-session.json path.
Checkpoint is missing	Run /checkpoint manually after confirming the current project folder. Auto checkpoints are created only after successful write phases.

Status messages to expect

Trackless structure inferred - soft generation hint only.

Non-browser execution mode - Vite/browser preview pipeline skipped.

Runtime will be decided by .builderstudio/project-contract.json.

Checkpoint saved - continuation can resume from disk.