ChatGPT

Visual Studio Code Settings (2025)

VS Code uses JSON files for all user/workspace settings. A typical setup includes a `.vscode/settings.json` (editor preferences), `keybindings.json` (custom shortcuts), `extensions.json` (recommended extensions), and `launch.json` (debug configs). For example, a Code-Bridge study shows configuring `"settingsSync.keybindingsPerPlatform": false` and ignoring certain extensions for sync [1]. Common settings include editor formatting (e.g. `"editor.formatOnSave": true`), font and tab size, theming, and language-specific defaults [2]. You can also enable AI tools: e.g. `"github.copilot.enable": { "*": true }` allows Copilot suggestions by default [3]. Use `extensions.json` to list extensions your project needs (e.g. Docker, Python, GitLens). Breakpoints and debug targets go in `launch.json` – for instance a Node launch config looks like `{ "version": "0.2.0", "configurations": [ { "type":"node", "request":"launch", "program":"${workspaceFolder}/app.js" } ] }` [4]. All these files can be version-controlled for sharing.

VS Code now has **built-in Settings Sync** to propagate your `settings.json`, `keybindings.json`, themes, and extensions via your GitHub/Microsoft account [5]. (For example, it can sync settings **across platforms** by disabling `settingsSync.keybindingsPerPlatform` [1].) Alternative tools like Crosside Sync can even synchronize settings locally between VS Code *and its forks (Cursor, VSCodium)* [6]. In practice, one can add key-value overrides and plugin settings via JSON as shown above, and rely on Sync or a dotfile manager for cross-machine consistency.

Popular Extensions and Plugins: Modern VS Code setups often include productivity/devops extensions. For example, many devs install ESLint/Prettier for formatting, GitLens for git insights, Docker/Kubernetes plugins, and AI assistants (GitHub Copilot, CodeWhisperer, etc). You can declare them in `extensions.json` so team members get prompted to install them. Custom settings for these plugins also live in `settings.json` (e.g. disabling telemetry or enabling auto-format). (See the Cursor-config gist [2] for a sample VS Code preferences JSON containing many editor and AI-tool settings.)

Sync & Portability: According to recent research, VSCode's cloud-backed Settings Sync is considered mature (alongside JetBrains' Backup & Sync) [7]. For cross-platform portability, DevContainers (via `devcontainer.json`) or GitHub Codespaces are now standard for reproducing the same IDE environment on any machine [7]. For example, one can build a Docker-based devcontainer that automatically installs the same VSCode extensions and applies the `settings.json` so your "mobile, laptop, desktop" setups match exactly. In a NixOS-based stack, the Nix Home Manager is often used to declaratively manage VS Code and other dotfiles [8] (ensuring identical config across machines). For instance, Home Manager can install VSCode extensions and even inject your `settings.json` into the environment at build time.

Cursor IDE (VSCode fork)

Cursor is an AI-powered IDE built on VS Code. It **uses a SQLite state file** (`state.vscdb`) instead of directly writing JSON settings [9]. However, since Cursor is essentially VSCode under the hood, you can still

apply the same `settings.json` file (e.g. in a `.vscode/` folder) to configure editor behavior. In fact, community tips note that placing a standard VS Code `settings.json` beside a Cursor workspace will "get recognized on project level."

In practice, Cursor users often maintain a VSCode-style settings file. For example, a public gist demonstrates a Cursor `settings.json` with common DevOps/dev preferences: it sets `"editor.fontSize": 14`, `"editor.tabSize": 2`, enables `formatOnSave` and `formatOnType`, auto-wrap, etc [2]. It also configures color themes (via the Peacock plugin favorites) and default formatters for JSON/TS/JS [2] [10]. Keybindings are similarly set (e.g. Vim-mode mappings under `vim.normalModeKeyBindings` [10]). Notably, the example enables GitHub Copilot in most filetypes via `"github.copilot.enable": { "*": true, "plaintext": true, ... }` [3] and turns off AWS CodeWhisperer. These illustrate how Cursor (like VS Code) can use JSON to turn on coding assistants and developer tools.

To keep settings synced between VSCode and Cursor, you can use the Crosside Sync extension (mentioned above) [6]. In other words, any `settings.json` tweaks you make in VSCode (plugins enabled, editor prefs, AI features) can be mirrored in Cursor. The gist settings above shows how a "DevOps vibe" profile might look: enabling live formatting, ignoring certain file deletions, customizing the terminal shell, and even reusing the VSCode Vim extension in Cursor.

LazyVim / Neovim

LazyVim is a modern Neovim distribution (written in Lua), so it does **not use JSON** for its core config – it relies on `init.lua` and plugin config. There's no single `settings.json` for LazyVim itself. Instead, users manage settings via Lua plugin files under `.config/nvim/`. (Popular plugin managers like `lazy.nvim` or distributions like NvChad/AstroNvim provide opinionated defaults.)

However, if you want VS Code to behave like LazyVim, you can mimic its keybindings. For example, a user posted a VSCode `settings.json` that maps LazyVim-style shortcuts into VSCode's Vim extension. This file sets `"vim.leader": "<space>"`, enables EasyMotion, surround, etc., and remaps `<C-h/j/k/l>` to move between panes [11]. It also binds `<leader>q` to close the window, `<leader>gg` to open Git, and so on. In short, by customizing VSCode's `vim.*` settings (as in the gist [11]), you can approximate LazyVim bindings.

More generally, LazyVim users do share configurations on GitHub (as Lua), but they can also work in VSCode. For instance, the official LazyVim docs note you can enable extra plugins for VSCode integration. And projects like *jetbrains-vscode* can convert JetBrains run/debug configs to VSCode (see Cross-IDE tools [12] in Code-Bridge).

Cross-Platform & DevOps Environments

In a DevOps/agentic robotics workflow, consistency across hardware is key. Modern practice (as of 2025) favors **containerized dev environments** and declarative dotfiles. Using Docker-based DevContainers or GitHub Codespaces lets you define a `devcontainer.json` with all tools (VSCode, languages, libraries) so that mobile devices, laptops, or cloud VMs run the same setup [7]. The Code-Bridge research highlights that DevContainers are "becoming the standard for reproducible environments" by 2025 [7]. It

recommends keeping the environment spec (tool versions, libraries) under version control so every machine spins up an identical IDE environment.

For OS-level consistency, **NixOS + Home Manager** is often used. In that model, all your VSCode settings, plugins, and even `settings.json` can be managed in Nix, guaranteeing the same configuration on any NixOS host [8]. For example, you can declare VSCode extensions and themes in Home Manager and have it write your settings file. This fits well with NixOS's reproducible goals in distributed computing.

On the shell side, using a cross-platform shell like Nushell can unify environments (Nushell configs are TOML/JSON-style as well). While not JSON-based for IDE settings, Nushell can be configured declaratively and paired with the IDE: VSCode's integrated terminal can be set to use Nushell (via `"terminal.integrated.defaultProfile.linux": "nu"` or similar in `settings.json`).

Summary: Use VSCode's JSON configs for all editor prefs and share them via Git or Settings Sync. Leverage tools like DevContainers and Nix/Home Manager for cross-machine sync [7] [8]. For the Cursor IDE, apply the same JSON settings (using a `.vscode` folder) and sync via Crosside [6]. For LazyVim, configure in Lua, but you can export similar keymaps to VSCode's settings (as shown in [62]). The examples above (especially the published gists and guides) illustrate up-to-date, advanced configs covering editors, plugins, and workflows [7] [2] [10] [11].

Sources: Examples and best practices are drawn from recent GitHub projects and research (e.g. the Code-Bridge "Dev Environment Sync" report [7] [1] [6] [8], public gists for Cursor and LazyVim configs [2] [10] [11]). These show how JSON settings are used in practice for VSCode, Cursor, and vim/VSCode integration.

---

[1] [4] [5] [6] [7] [8] [9] [12] DEV_ENVIRONMENT_SYNC_RESEARCH_2025.md
https://github.com/TriAlliance/Code-bridge/blob/f0bb24edcc06a96cd0975f963bce599583ef6e23/DEV_ENVIRONMENT_SYNC_RESEARCH_2025.md

[2] [3] [10] settings.json for Cursor(VSCode) · GitHub
https://gist.github.com/tomoima525/034c7f4360f238bbbdfe5b037937dcd6

[11] LazyVim keymappings for VSCode · GitHub
https://gist.github.com/brotherkaif/098f14950d538cb7001a7e04d8d884ee