

One server per extension

Lab note 6 15 Oct 2026 6 min read
written by Claude from this repository's data

lsp-plus answers Claude's code searches from a language server. That only works where a server answers. The obvious next question, asked by someone who works with infrastructure all day: what about Terraform, Ansible, Kubernetes YAML? This note is about adding languages, and how often "there is a language server for that" turned out to be the start of the work, not the end.

The rule that shapes everything

Claude Code picks a language server by file extension, and only by extension: no file patterns, no first-line detection. Each extension gets exactly one server; when two plugins claim the same one, the first registered wins (plugin docs). For most languages that is fine. For `.yaml` it is a problem, because Kubernetes manifests, Compose files, GitHub Actions, Ansible playbooks and Helm values all share it.

Five languages, five surprises

We gave lsp-plus Go, Kotlin and Ruby (whose servers it could install but whose files its grep answer ignored), and declared servers for shell scripts and Terraform. Then we tested each against a real server, in a real Claude Code session, on a small repository where we knew the right answer.

- **Go** worked at once. gopls even found callers through method receivers (`func (c *Calculator)`
`Total`).
- **Shell** worked after one change. bash-language-server has no call hierarchy, so a search for a function got no answer. lsp-plus now falls back to the server's references when there is no call hierarchy.
- **Ruby** crashed with exit code 78 before answering anything. ruby-lsp refuses to start in a project with a `Gemfile` but no `Gemfile.lock`. Most apps commit the lock file; many gems do not. Now in the setup guide.
- **Terraform** answered symbol searches but found no references. Our own LSP client got the same, so it was the server, not Claude Code. Its log said why: no provider schema for `hashicorp/aws`. terraform-ls only sees references inside blocks whose schema it knows, and the Homebrew build of terraform-ls ships without the provider schemas HashiCorp's own build embeds (0 schema entries in the binary against 6). Installed from HashiCorp's tap, every `var.region` was found.
- **Kotlin** we could not test: JetBrains' Homebrew formula pointed at a download that returned 404 that day. It is covered by tests only.

Terraform also needed lsp-plus to understand its names. A grep for `var.region` or `aws_s3_bucket.logs` searches for an address, not an identifier. lsp-plus now maps an address to the block that declares it, asks the server in each module (directory) where the grep found it, and answers with the references:

```
'var.region' per the language server: 4 references in code, in 3 files (code only: st
main.tf: 2
outputs.tf: 2, 6
providers.tf: 2
```

Ansible and Kubernetes: no references to give

For YAML the research came first, and it was sobering (sources):

- The **Ansible language server** offers hover, completion, definition and diagnostics, but no references and no symbols; its "definition" jumps from a module name to the module's Python source. A pull request adding references for handlers, variables and roles was closed unmerged. And it decides a file is Ansible by path patterns and content, which Claude Code cannot express.
- **yaml-language-server** with Kubernetes schemas validates structure (a misspelled field, a string where a number belongs) but has no references: its "definition" only jumps from a YAML alias to its anchor in the same file.
- No language server we found resolves the names that actually connect a Kubernetes deployment: a Service's selector to the pods it selects, a `configMapRef` to the ConfigMap, a RoleBinding to its ServiceAccount.

So the question became: one YAML server for everything is forced on us anyway; could it also check the names?

yaml-plus

yaml-plus is one server for every `.yaml` file. It runs yaml-language-server underneath for syntax and schemas, choosing the schema from each file's content (Kubernetes by `apiVersion` and `kind`, Ansible by structure). It skips Helm templates, which are not YAML until rendered. On top it indexes the repository's names and checks the references between them:

- **Kubernetes:** ConfigMaps, ServiceAccounts and volume claims a pod uses; Services and the workloads they select; Ingress backends, autoscaler targets, role bindings; kustomize paths.
- **Compose:** `depends_on`, networks, volumes, configs, secrets.
- **GitHub Actions:** `needs`, `needs.X.outputs`, `steps.X`, inputs, local actions.
- **Ansible:** handlers and `notify`, roles, included task files.

It also answers go-to-definition and find-references for those names.

Its rule is to say nothing it is not sure of. It never flags a Secret (they are often created outside the repository), never a name from a remote base, never a variable. Run over about 30 local repositories and 6 public ones (Argo CD, kube-prometheus, the Kubernetes examples, awesome-compose, an Ansible book, VS Code), it raised one warning in 5,041 name uses: a PersistentVolumeClaim whose manifest exists only inside a README. It indexes 4,000 manifests in about 0.7 s.

One thing we did not expect. In a live session, yaml-plus's warning reached Claude after an edit:

ConfigMap "cart-config" (namespace shop) is not declared in this repository (did you mean "cart-config"?)

Reading the same broken file delivered nothing. Claude Code syncs a file to its language server when it is edited or asked about, not when it is read. A checker that only speaks after edits cannot help Claude *find* a broken reference, only avoid making one. yaml-plus therefore gained hooks: a read of a YAML file gets that file's name problems, and a grep for a name that hits YAML files gets where it is declared and used, the way lsp-plus answers code searches.

What it measured

We generated a 54-file deployment repository (ten services across Kubernetes, Compose, two GitHub Actions workflows and six Ansible roles) and four tasks:

- a ConfigMap rename across 11 references;
- six planted broken references, one in each kind of file and two more in Kubernetes;
- "which objects refer to this ServiceAccount";
- a Compose change.

Every task was graded by parsing the YAML, never by the server under test.

Model	yaml-plus added to the pack	95% interval	Runs	Correct
Opus 5.5	+2.0% cost	-1.3% to +5.3%	64	all
Haiku 5.5	about the same, at roughly one cent a run		32	all

yaml-plus spoke 56 times in Opus's runs. Opus found all six broken references and completed the 11-reference rename without it. We expected Haiku, a weaker model, to miss what the checker catches; it missed nothing either.

Fifty-four files are few enough for any model to read whole, and a checker cannot save what a model never misses. The question `yaml-plus` exists for is different: do names resolve correctly every time, and faster, in a repository too big to read?

So we made one. The same generator with 300 more services comes to 954 YAML files, and we hid ten broken references in it: the six from before and four among the generated services (a misspelled `ConfigMap`, a `ServiceAccount` spelled with the letter O for a zero, a `Service` whose selector matches no pods, an autoscaler aimed at a `Deployment` that does not exist).

Model	Without <code>yaml-plus</code>	With <code>yaml-plus</code>
Opus 5.5	10 of 10 in every run, \$0.61, 10.2 requests	10 of 10 in every run, \$0.70, 10.0 requests
Haiku 5.5	10 of 10 in every run, \$0.03	10 of 10 in three runs, 8 in one, \$0.03

Four runs per cell. Opus did not read 954 files. It listed them, read one of each kind, and then **wrote its own reference checker**: about 50 lines of Python with `PyYAML`, which it ran over the whole repository, and confirmed the result with `kubectl kustomize`. In one step, it built for this task the tool we had built for every task.

That is the honest end of this experiment. Within the time we could give it, we did not find the repository where `yaml-plus` earns its place: small ones are read whole, large ones get a checker written on the spot. The server costs no tokens until it speaks, only memory (about 200 MB for 4,000 manifests, plus `yaml-language-server`), so it does no harm to keep. But nothing here says it helps, and by our rule that keeps it out of the pack. A real infrastructure repository with irregular names, Helm charts and `kustomize` overlays, too tangled for a 50-line script, is the test it would still need.

Method & limits

Language checks: one headless Claude Code session per language on a small repository with known answers, the hook's note read from the transcript; `terraform-ls` also queried directly with a minimal LSP client. `yaml-plus`: 44 server tests and 13 hook tests, and the false-positive sweep described above, run locally. The YAML benchmark's generator and grader are in `yaml_repo.py`; runs in `results-yaml.jsonl`, `results-yaml-b.jsonl` and `results-yaml-haiku.jsonl`; the 954-file repository in `results-yaml-big.jsonl` and `results-yaml-big-haiku.jsonl`.

Sources

- Claude Code, Plugin components: LSP servers.
- HashiCorp, `terraform-ls`, `docs/schema.md` (provider schemas bundled at compile time in HashiCorp's released binaries).

- Red Hat, ansible-language-server, and the closed pull request #2647.
- Red Hat, yaml-language-server.
- Ruby LSP; bash-language-server; gopls.