Vault Dev Server — Hands-On Cheat Sheet

Scope. A concise, practical guide for learning and testing HashiCorp Vault using the *dev server*. This mode is for local exploration only.

At a Glance

- **Dev mode is not production**: runs without TLS, keeps data in memory, starts unsealed, prints a single unseal key and a root token to the console.
- Default API address: `http://127.0.0.1:8200`.
- Use the `VAULT_ADDR` environment variable for the CLI.

Prerequisites

- Install `vault` CLI and server from HashiCorp.
- (Recommended) Install `jq` for JSON output filtering.
- If compiling this PDF: `minted` requires `-shell-escape`.

Start the Dev Server

Run Vault (Terminal 1)

```
vault server -dev
```

Notes:

- The console prints an **Unseal Key** and a **Root Token**. In dev mode the server starts unsealed.
- Leave this terminal running; it is the server process.

Configure the CLI (Terminal 2)

```
export VAULT_ADDR="http://127.0.0.1:8200"
vault status
```

Your First Secret (KV)

Write and Read (human-friendly)

```
vault kv put secret/mySecret password=myPassword
vault kv get secret/mySecret
```

Read as JSON (for scripts)

```
vault kv get -format=json secret/mySecret | jq
```

Secrets Engines Basics

Discover and Enable

```
# List current secrets engines
vault secrets list

# Enable another KV mount at a custom path
vault secrets enable -path=myapp kv

# Use the new mount
vault kv put myapp/myOtherSecret key=value
vault kv get myapp/myOtherSecret
```

Explore Paths and Verbs

```
# Show path patterns and help for an engine
vault path-help secret

# Example: enable a database engine (demo)
vault secrets enable database
vault path-help database
vault path-help database/roles
```

Authentication: Userpass Demo

Enable and Inspect Auth Methods

```
vault auth list
vault auth enable userpass
```

Create a User and Log In

```
# Create a local user
vault write auth/userpass/users/vaultuser password=vault

# Login with userpass method (generates a client token)
vault login -method=userpass username=vaultuser password=vault

# Or login with the dev server's root token from Terminal 1
vault login <ROOT_TOKEN>
```

Tokens: Create, Inspect, Revoke

```
# Mint a child token (inherits parent's policies)
vault token create

# List token accessors (manage tokens without seeing their secrets)
vault list auth/token/accessors

# Lookup / revoke using an accessor
vault token lookup -accessor <ACCESSOR>
vault token revoke -accessor <ACCESSOR>

# Short-lived token
vault token create --ttl="5m"
```

Policies: Authorization in Vault

Capabilities: `create`, `read`, `update`, `delete`, `list`, plus `sudo` and `deny`. If any attached policy denies an action, **deny wins**.

Inspect and Upload Policies

```
# What policies exist?
vault policy list
vault policy read default

# Upload policies from local files
vault policy write dev-policy dev-policy.hcl
vault policy write app-policy app-policy.hcl

# Attach policies to userpass users
vault write auth/userpass/users/dev  password=dev  policies=dev-policy
vault write auth/userpass/users/app  password=app  policies=app-policy
```

Check Effective Capabilities

```
# KV v2 data path format is /data/<path>
vault token capabilities secret/data/dev/
```

Exercise the Policies

```
vault kv put secret/dev/appsecret user=dbuser
vault kv get secret/dev/appsecret
```

Environment and CLI Tips

```
# Point CLI to the server
export VAULT_ADDR="http://127.0.0.1:8200"

# Temporarily use a specific token
export VAULT_TOKEN="<CLIENT_OR_ROOT_TOKEN>"

# Verify who you are
vault token lookup
```

Troubleshooting & Gotchas

- **Cannot connect**: Ensure the server (Terminal 1) is running and `VAULT_ADDR` matches the printed address.
- **Permission denied**: Check your token and attached policies. Use `vault token capabilities <path>` to diagnose.
- **KV v1 vs v2 paths**: KV v2 uses `/data/` and `/metadata/` API paths internally; CLI subcommands handle this, but capabilities checks should use `secret/data/...`.
- **Dev mode data loss**: All data is in-memory. Restarting clears everything.
- **Minted compilation**: If building this PDF, compile with `-shell-escape`.

Production-Oriented Notes (for Later)

- Use TLS, a durable storage backend, and sealed starts; do not use dev mode in production.
- Prefer real identity-backed auth (GitHub, LDAP/AD, AWS, Kubernetes) over `userpass`.
- Rotate tokens; use short TTLs and renewals where appropriate.
- Practice least privilege; **deny** should be explicit for sensitive paths.

One-Page Quick Reference

```
# Start (T1)
vault server -dev

# Configure CLI (T2)
export VAULT_ADDR="http://127.0.0.1:8200"
vault status

# KV
vault kv put secret/mySecret password=myPassword
vault kv get secret/mySecret
vault kv get -format=json secret/mySecret | jq

# Engines
vault secrets list
vault secrets enable -path=myapp kv
vault kv put myapp/myOtherSecret key=value
vault kv get myapp/myOtherSecret
vault path-help secret

# Auth (userpass)
vault auth list
vault auth enable userpass
vault write auth/userpass/users/vaultuser password=vault
vault login -method=userpass username=vaultuser password=vault
vault login <ROOT_TOKEN>

# Tokens
vault token create
vault list auth/token/accessors
vault token lookup -accessor <ACCESSOR>
vault token revoke -accessor <ACCESSOR>
vault token create --ttl="5m"

# Policies
vault policy list
vault policy read default
vault policy write dev-policy dev-policy.hcl
vault write auth/userpass/users/dev  password=dev  policies=dev-policy
vault token capabilities secret/data/dev/
vault kv put secret/dev/appsecret user=dbuser
vault kv get secret/dev/appsecret
```