RAD Deployment and Development (In Practice)

This document describes the *concrete deployment and development tooling* that implements the RAD framework today. It lives alongside the product vision (Megagit, Kinetic, Drift, ARC, etc.) and the concept papers — the vision defines *what* we're building toward; this describes *what is already running* and how it fits the same ecosystem.

Overview

The RAD framework in practice provides:

- *Lightning-fast deployment pipelines* for multiple application types (web backend, Flutter web/desktop, static sites, WordPress, Linux tools), with minimal or no dedicated testing stages for rapid iteration.
- A *unified CLI* (`rad`) for development host setup, dev boxes (Distrobox), cloud components, and application build/deploy.
- *Core services*: K3s (single-node Kubernetes), Podman registry, FTP server for binaries, NGINX ingress.
- *Templates* for new projects so new apps can be added by copying a template and adjusting.

This aligns with the product vision of a "loosely-opinionated development pipeline" and "next-gen unified package management and deployment." Megagit (multi-repo Git) and Kinetic (CI/CD) will eventually sit on top of or alongside this deployment layer; Drift will configure the machines that run it.

The `rad` CLI

The *rad* script is the main entry point. It organizes operations into domains:

Domain	Purpose
dev-host	Development host configuration: system directories, PATH, base tooling (e.g. Podman, base-dev packages).
dev-box	Distrobox development containers: build, enter, stop, remove. Each box is defined by a `distrobox.ini` and optional setup script.
cloud	Cloud components (e.g. static-website, web-backend): build and deploy.
app	Application templates (e.g. flutter-web, linux-tool): build and deploy to Kubernetes.

Example commands

```
rad dev-host setupnrad dev-box build embeddednrad dev-box build –allnrad dev-box enter embeddednrad cloud build static-websitenrad cloud deploy –allnrad app build flutter-webnrad app deploy –all
```

Configuration tooling (Ansible-style modules, system-config, packages) is loaded from `configuration-tooling` and used by `rad dev-host setup` and by Distrobox setup scripts. So the same "configure the machine" philosophy that will appear in *Drift* is already present in the dev-host and dev-box layers.

Architecture (Current)

- *K3s* — Single-node Kubernetes for container orchestration.
- *Podman Registry* — Local container registry (e.g. port 5000) for image storage.
- *FTP Server* — Hosting for application binaries and packages (e.g. Linux tools, Flutter desktop builds).
- *NGINX Ingress* — Web traffic routing to services.

Deployments typically: build (e.g. `podman build` or `cargo build` / `flutter build`), push to registry or upload to FTP, then apply Kubernetes manifests or make binaries available for download.

Development System

Development is intentionally *offline-capable* and separate from the cloud:

- *Dev-host* — One-time or occasional setup: install tooling, create `~/.system`-style directories, configure Podman.
- *Dev-boxes* — Per-project Distrobox containers that consume base images and project-specific `distrobox.ini` + setup scripts. Developers work inside these containers with access to host HOME, graphics, and network.
- *Configuration tooling* — Reusable scripts (language runtimes, IDEs, containerization) that both the host and the boxes can use.

This mirrors the idea of "developer environment (simplicity collapse zone)" in the Kinetic Vector diagram: local development can collapse to a single machine and CLI while production uses the full Kinetic/ARC pipeline.

Relationship to the Product Roadmap

Vision component	Current implementation / bridge
Megagit	Not yet integrated. Multi-repo operations (e.g. `gitr sync` across repos) can be used from the same host or dev-box that runs `rad`.
Kinetic	Conceptual match: "Forge" (build/test), push, ARC, delivery. Today: ad-hoc `deploy.sh` scripts and K3s; future: Kinetic pipelines and workers.
ARC	Registry and artifact storage today (Podman registry, FTP). ARC Framework will standardize artifact classification and storage.
Drift	Configuration tooling (Ansible playbooks, shell modules) and `rad dev-host` / dev-box setup are precursors to a declarative, Drift-style system definition.
Registry One	Application hosting is currently K3s + NGINX; Registry One will formalize hosting in the RAD story.

Diagrams (In This Portal)

- RAD Diagram — High-level RAD overview.
- RAD Framework Tooling — Tooling and layout.
- Kinetic Vector Diagram — Full Kinetic lifecycle: Developer Environment → Kinetic Forge (Build & Test) → ARC (The Ark) → Delivery Vectors → Post Server.
- The Simplicity Rule — How production (build workers, ARC, Kinetic backend) "collapses" to local (kinetic CLI, local filesystem / `.arc-cache`) for development.

These diagrams were created in the AppDevelopmentFramework tooling repository and are included here for the product portal.

Summary

RAD in practice is already a working deployment and development framework: *rad* CLI, K3s, registry, FTP, Distrobox-based dev boxes, and configuration tooling. The product vision (Megagit, Kinetic, Drift, ARC, Registry One, Prism, Insanity) extends this with dedicated tools for Git, CI/CD, system definition, artifacts, and hosting. Documentation for those components lives in this portal under RAD Framework Overview and the individual product sections.