A TOOL FOR MANAGING YOUR DEVOPS INFRASTRUCTURE
What is Unfurl?

A command-line tool that:

1. **Deploys** applications using a high-level declarative vocabulary so you can describe your infrastructure independent of cloud provider details.

2. **Manages** your deployments (configuration, tools, history, state) in Git.
THE BASICS:

- Self-contained command-line tool that runs locally: no server or agent software involved
- All state is saved in user-editable YAML configuration files: no database
- Manages git repositories for configuration and artifacts

- OPEN SOURCE
- COMMUNITY-DRIVEN
- DECENTRALIZED (VIA GIT)
- STANDARDS- BASED (TOSCA)
Unfurl tracks your:

- Configuration
- Secrets
- Code dependencies
- Software version
- Deployment history

... all in git!
Works with your tools

Includes out-of-the-box support for

- Terraform
- Ansible
- Shell commands
- Helm
- OctoDNS
- Kubernetes
- Docker
- Supervisor
Motivation: Imagine if you could freely reuse, modify, and fork live services just like open source code.
Why Use Unfurl?

**SIMPLICITY:**
Hide complexity, no need to be a DevOps guru

**FLEXIBILITY:**
Easily change infra as your needs grow

**NO LOCK-IN:**
Open source
All data stored in Git
Cloud Independence
No single DevOps tool provides a complete solution, coding is needed to integrate.

<table>
<thead>
<tr>
<th></th>
<th>Helm</th>
<th>Terraform/Pulumi</th>
<th>Ansible</th>
<th>Chef/Puppet/Salt</th>
<th>CI/CD/Gitops</th>
<th>Docker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packaging</td>
<td>Kubernetes only</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Minimal</td>
</tr>
<tr>
<td>Provisioning</td>
<td>No</td>
<td>Coding</td>
<td>Lo-code</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Day two Operations</td>
<td>No</td>
<td>No</td>
<td>Lo-code</td>
<td>Coding</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Change Management</td>
<td>Kubernetes only</td>
<td>Enterprise-only</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
DEMO
Unfurl Processing Model

TOSCA Spec

service-template.yaml

Local config + secrets

ensemble.yaml

unfurl.yaml

“unfurl deploy”

Step 1: delegate to isolated runtime

Execute Plan

Step 2: invoke configurators

Record Results

Step 3: update status and commit changes to git
<table>
<thead>
<tr>
<th>Context</th>
<th>Spec</th>
<th>Status</th>
<th>Lock</th>
</tr>
</thead>
</table>
| **Environments:** Describes the environment that the Unfurl deployment process runs in, e.g.  
- Tools and repository locations  
- Credentials | **Spec:** TOSCA service template and explicitly declared instances | **Status:** Deployed instances and their current status | **Lock:** Record precise digests and versions of the artifacts and repositories used during deployment. |
Unfurl allows you to collect your configuration information and organize them by environment.
Spec: Model Your Cloud Infrastructure

```yaml
spec:
  service_template:
    topology_template:
      inputs:
        rootdomain:
          type: string
          default: unfurl.run
```
Combine & connect cloud-independent building blocks.

Deploy implementations that match your infrastructure and configuration.
What is TOSCA?

- Used to manage applications in cloud and telecom network management
- **YAML vocabulary** that describes:
  - The architecture of a cloud application or service
  - The infrastructure the application requires in order to operate
  - The operations for deploying and managing the application

**KEY CONCEPTS:**
- Nodes
- Operations
- Relationships
- Requirements
- Capabilities
- Configurations
- Policies
TOSCA DESCRIBES THE TOPOLOGY OF THE DEPLOYMENT OF CLOUD APPLICATIONS AND SERVICES

**Nodes** - are the resources or components that will be materialized or consumed in the deployment topology.

**Topologies**

- **Source node**: Node_Type_A
- **Target node**: Node_Type_B

**Requirement** - **ConnectsTo** - **Capability**

**Groups**

- Create Logical, Management or Policy groups (1 or more nodes)

**Relationship templates** to describe connections, dependencies, deployment ordering

**Node templates** to describe components in the topology structure

**Relationship templates** to describe requirements connecting components

**Requirement - Capability**

Relationships can be customized to match specific source requirements to target capabilities.
Portability – TOSCA Orchestrators find “Best Match” during deployment

TOSCA APPLICATIONS ARE PORTABLE TO DIFFERENT CLOUD INFRASTRUCTURES

By expressing application Requirements independently from cloud provider Capabilities

**TOSCA Orchestration**

**Automatic Matching & Optimization**

**Infrastructure Capabilities**

**Application Requirements**

**TOSCA Service Template**

- App
- DB
- Compute1
- Compute2
- Network
- Storage
- Scaling Policy

Cloud Provider A
Cloud Provider B
Cloud Provider C
status:
  inputs:
    rootdomain: unfurl.run
  outputs: {}
  instances:
    staging_site:
      template: staging_site
      readyState:
        local: ok
        state: created

lock:
  runtime:
    unfurl:
      version: 0.1.1.dev20
      digest: 0.1.0-29-gf41588a-dirty
    toolVersions:
      terraform:
        - 0.12.29
  repositories:
    - name: self
      url: git-local://5fd0f9...d0f8e80429:self/
      revision: 6c1706140dce9338182ac46f37d04b5397ce0723-dirty
Reproducible Ensembles

Reproducible Artifacts

Isolated Deployment Environment

Deterministic Deployment Process

Ensembles

Isolated Cloud Environment
Vision: A Free and Open Cloud
THANK YOU!

adam@onecommons.org
www.unfurl.run
github.com/onecommons/unfurl
www.onecommons.org