Learning Path: Mastering Contribution To Meshery

Certification: Contributing To Meshery

Innovation through Collaboration

Document Purpose

1. To provide a roadmap for a complete Learning Path to Mastering Contribution to Meshery.
2. To enable collaborative drafting and review of the content itself (written in markdown).

Tips and Tricks from Maintainers

Having Maintainers offer questions and review as beta testers.

Create FAQs

- What happens if I fail? Can I retake and how many times, if so?
- Does my credential (certificate) expire?
- How do I show-off my credentials (I'm proud of it)?
- Other faqs

Create one exam per architectural component.

- SErver, ui, cli,
- Adapters
-

Lee Calcote Create a badge graphics

Ritesh Karankal Brainstorm a name

Lee Calcote Create a new repo from template: github.com/meshery-extensions/meshery-academy

Blog post

sSOl
Put into an email and send

Document Scope

Out of Scope:

1. Regurgitating all of Meshery Docs.

In Scope:

1. Highlighting best practices, covering common user flows.

Resources:

- See CNCF Hub for a taxonomy of Learning Paths.

Draft Learning Path Table of Contents

Learning Path: Contributing to Meshery

Course: Overview

Chapter: Contributing overview

About Meshery

Chapter: Projects and Repository overview

Chapter: How, Why, and Where to Contribute

Section: How to Contribute

Assessing your skills & interests, choosing an area to contribute, Where to Ask for help, Discuss, and Collaborate: GitHub Issues, Slack, Discussion Forum, and Meetings.

Section: Why to Contribute

Learning opportunity, impact, recognition

Section: Where to Contribute

Documentation, Codebase, Community & Outreach, Design & UX, Testing and QA, Project Management

Chapter: Contributing Flow

Chapter: GitHub Process

Chapter: Community Guidelines and Code of Conduct

Course: Working with Docs

In this course, you will learn how Meshery docs are structured, how to set up your dev environment, and how to contribute docs that actually get merged.

Chapter: Meshery docs

Section: Meshery Docs structure

Overview of docs.meshery.io layout, How users can navigate, and find relevant content

Meshery Docs live at docs.meshery.io and act as the go-to place for users, contributors, and maintainers to learn, explore, and contribute to Meshery. Understanding how these docs are organized will help you navigate them smoothly and know where to add your contributions.

Layout Overview

- **Home**: The landing page introduces Meshery and links to the guides.
- **Guides and Tutorials:** Step-by-Step instructions for using Meshery features.
- **Concepts and Architecture:** Background material that explains Meshery's building blocks, like components, patterns, models, and more.
- **Installation:** Platform-specific installation instructions (Docker, Kubernetes, Minikube, etc.)
- **Reference:** Command-line reference (mesheryctl), API reference, and error codes.
- **Contributing:** Documentation for contributors, explaining workflows, style guides, and best practices for writing and submitting changes.

Navigation

- The sidebar is the main way to move through the docs. It is grouped by topics like *Installation, Guides, Concepts*, etc.
- Each page uses YAML frontmatter at the top to control how it shows up in the sidebar.
- Internal links (cross-links) make it easy to jump between related pages without losing context.

Why Structure Matters

Keeping the docs structured ensures:

- Clarity for users: Readers can quickly find what they are looking for.
- Ease for contributors: You know exactly where your new page or section belongs.
- Consistency: Docs look and feel uniform across all topics.

Chapter: Setting up the dev environment

A local docs setup lets you preview changes before you open a PR. Pick your OS and follow the steps. If something feels off, there is a small Troubleshooting box in each section.

Section: For Windows

Goal: run the Meshery docs site locally with Jekyll inside WSL2 (Linux on Windows).

Section: For Linux

Goal: Same as Windows, just native Linux.

1. Clone and install deps:
   `git clone https://github.com/<your-username>/meshery.git`
   `cd meshery/docs`

2. Serve
   `make docs`
   or
   `bundle exec jekyll serve --drafts --config _config_dev.yml`

Section: For MacOs

Goal: Run Jekyll with system Ruby or rbenv.

1. Install Homebrew (if needed)
   https://brew.sh

2. Install Ruby

3. Clone and install deps
   `git clone https://github.com/<your-username>/meshery.git`
   `cd meshery/docs`
4. Serve
   `make docs`
   or
   `bundle exec jekyll serve --drafts --config _config_dev.yml`

Section: Serving the site (what the commands do)

- make docs runs Jekyll in dev mode (drafts enabled, local server, live reload).
- bundle exec jekyll serve --drafts --config _config_dev.yml does the same, but explicitly (handy when make is not available or you want to toggle flags).

When it boots, open http://localhost:4000 and browse your changes.

Section: Using Docker

Best when you don't want to install Ruby/Jekyll locally.
From the meshery/docs folder:
`make docker`
This builds a container and serves the site. Open the port it prints (usually http://localhost:4000).
Heads-up: This path can be flaky on Windows. If it misbehaves, prefer WSL2 or Codespaces/Gitpod.

Section: Using Gitpod

Best when you want zero local setup.

1. Install the Gitpod browser extension.
2. Open your fork on GitHub and click Gitpod (button appears with the extension).
3. In the Gitpod terminal:
   `cd docs`
   `make docs`
   Gitpod will expose a preview URL in the Ports panel.

Section: Serve GitHub Codespaces

Best when you want a cloud VS Code with your fork.

1. On your fork, click **Code** → **Open with Codespaces** → **New codespace**.
2. In the terminal:
   `cd docs`
   `make docs`

Codespaces will show a forwarding URL (click it). If you want to work from local VS Code connected to your Codespace, you can just open the port link from there.

Chapter: Documentation Contribution Flow

Section: Contribute and Preview the changes

Contributing to Meshery Docs follows a simple and structured process to ensure consistency and quality.

1. **Fork the Repository**
   - Start by forking the Meshery repository into your GitHub account.
2. **Clone Locally**
   - Clone your fork to your local machine and navigate to the /docs directory.

     ```
     git clone https://github.com/<your-username>/meshery.git
     cd meshery/docs
     ```

3. **Create a New Branch**
   - Always create a new branch for your changes.
     `git checkout -b my-docs-update`
4. **Make Edits**
   - Update the relevant Markdown file (.md).
   - If you are adding a new page, don't forget to update navigation or sidebar files if needed.
5. **Preview Locally**
   - Run make docs (or the relevant command for your platform) to serve the documentation site locally.
   - Open `http://localhost:4000` in your browser to see the changes.
6. **Commit with Sign-off**
   - Meshery uses the Developer Certificate of Origin (DCO). Always sign your commits.
     `git commit -s -m "docs: updated contributing guide"`

7. **Push and Open a Pull Request**
   - Push your branch to your fork and open a PR against `meshery/meshery:master`.
   - A Netlify preview will be automatically generated for reviewers to test.

This flow ensures all the contributions are traceable, reviewable, and meet project standards.

Chapter: Documentation Framework

The Meshery Docs site is powered by modern static site tooling. Understanding the framework helps contributors work effectively with the system.

Section: What Powers the doc - rendering and structuring

- **YAML Frontmatter**

Each documentation file begins with a block of YAML frontmatter. This metadata defines properties of the page:

—
layout: default
title: "Contributing to Meshery Docs"
abstract: "Learn how to contribute to Meshery Docs."
permalink: guides/tutorials/contributing-to-docs
language: en

- **layout** → Which template to use (default = standard page look).
- **title** → The name of your page (shows up in headings & navigation).
- **abstract** → A summary that appears in previews or cards.
- **permalink** → The URL path where this page will live (e.g., `/guides/tutorials/contributing-to-docs`).
- **language** → The language of the page (default is en).

➢ **TOC in sidebar**
Controlled by front matter (e.g., toc: true) and sidebar config. Don't hand-roll complex sidebars in-page; follow existing patterns.

➢ **Liquid**
Meshery Docs use Liquid, the templating language from Jekyll, to dynamically include content, variables, and layouts.

- Example: {% include alert.html type="info" title="Heads up!" %}
  This displays a styled alert box without repeating code.

Jekyll templating syntax

➢ **Jekyll**

- Jekyll is a static site generator used to build Meshery Docs.
- Converts Markdown files (.md) into static HTML pages.
- Applies layouts and themes for consistent styling.
- Supports drafts, collections, and custom plugins.

➢ **Theme** https://github.com/vsoch/docsy-jekyll

- Meshery Docs are styled with the Docsy Jekyll Theme.
- Ensures responsive design and clean navigation.
- Simplifies maintenance and reduces custom CSS.

Chapter: Understanding the Flow of Meshery Docs Rendering

Section: Flow

Meshery Docs are written in **Markdown (.md)**, which are then processed through **Jekyll** and rendered into static **HTML pages**. The flow is:
Markdown (.md) → Jekyll (Liquid + Layouts) → HTML → Deployed site

- **Markdown** provides the content.

- **YAML frontmatter** adds metadata like title, description, and layout.
- **Jekyll with Liquid templates** applies layouts and formatting.
- **Docsy Jekyll theme** ensures consistent styling and navigation.

  Understanding this flow helps contributors preview how their raw `.md` edits become styled pages on docs.meshery.io.

Chapter: Using the Features of Meshery docs

Section: Clipboard

Code snippets in Meshery Docs automatically include a clipboard copy button.

```
mesheryctl system start
```

```
Readers can copy commands with one click, improving usability.

### Section: Alerts
Use alerts to highlight important information. Meshery Docs includes reusable
`alert.html` partials.
```liquid
{% include alert.html type="info" title="Note" %}
```

Types include: `info`, `warning`, `danger`, `success`, `primary`, etc.

Section: Alerts

Alerts are used in Meshery Docs to call attention to important notes, warnings, or tips. They help readers quickly spot key information without cluttering the flow of the page.

Meshery Docs provides a reusable include file, `alert.html`, for creating alerts. You can choose from different types such as `info`, `warning`, `success`, `danger`, and more.

Example (info alert):

```
{% include alert.html type="info" title="Heads up!" %}
This is an informational message.
{% endinclude %}
```

This will render a styled alert box with the heading “Heads up!” and your message beneath it.

Other alert types available:

- `info` – highlights helpful information.
- `warning` – flags something users should be careful about.
- `success` – indicates an action completed successfully.
```

- `danger` – calls out critical issues or errors.

Section: Image Handling

Images can be added with Markdown:

```
![Alt text](/assets/img/example.png)
```

Or with custom HTML for controlling size:

```
<img src="/assets/img/example.png" style="width:500px;" alt="Example">
```

Section: Quotes

Use > for blockquotes:

```
> This is an example of a blockquote.
```

Chapter: Table of Contents in Sidebar

Section: Manual generation of TOC in Meshery Docs

Meshery Docs sidebar navigation is built from the **TOC (table of contents)** in YAML. Example:

```
toc:
- title: Docs
  subfolderitems:
  - page: Setup
    url: /project/contributing/contributing-docs
```

TOC ensures chapters and sections appear in the sidebar hierarchy (parent → child → grandchild).

Chapter: How Docs are Versioned

Section: Meshery maintains multiple versions of documentation. This ensures users can reference docs matching the Meshery release they're running. Contributors may need to:

- Add content under the current version folder.
- Update links and redirects for older versions if the content moves.
- Ensure breaking changes in Meshery are documented clearly in the right version.

Chapter: Writing Tips and Resources

Section: <section>

- **Clarity first**: Write as if explaining to a newcomer.
- **Keep it consistent**: Follow the tone/style already used in Meshery Docs.
- **Use active voice**: e.g., "Run `make docs` to serve the site."
- **Link generously**: If content is explained elsewhere (e.g., tutorials, CLI guides), link to it.
- **Test locally**: Always preview your edits before PR.

  Resources:
  Jekyll Docs
  Liquid Templating
  Meshery Docs Contributing Guide

Tutorial: Contributor Training Series: Working with Meshery Docs (Zihan Kuang)

Course: Contributing to Meshery UI

Chapter: Meshery UI

Section: Overview

Section: Architecture

ReactJS, NextJS, Material UI, BillboardJS, CytoscapeJS, Redux Toolkit, Sistent, Schemas

Section: Design

Userflow/Wireframing/Mockups - Figma files
Design Goals

Section: Setup Development Environment

Linting, install UI dependencies, build and export UI, Run Meshery server, UI development server

Section: Meshery server API's

REST API, GraphQL API

Quiz

Chapter: Schema-Driven UI Development

Section: Overview

Section: Repo Structure

Section: Schema Drive UI Development Workflow

- Define or update the schema
- Generate TS types and Schema objects
- Build and export
- Use schema

Section: Integration Points in UI

- RJSF
- General Form UI
- UI-Specific Description
- Type Safety
  Chapter: Sistent Design System

Quiz

-

Section: Intro to Notification Center

Chapter: Dashboard Widgets

Section: Notification Center overview

Quiz

Chapter: Notification Center

Section: What is a Notification Center

Section: Metadata Formatter

Section: How Notification Metadata is Rendered

Section: Repository Structure

NotificationCenter/ (Root Directory)
formatters/ (NotificationCenter/formatters)

Section: Types of Event Specific Notification Formatter

- Common Formatter
- Error Formatter
- Model Registry Formatter
- Relation Evaluation Formatter
- Dry Run Formatter
- Deployment summary Formatter
- Property Formatters and Property Link Formatters

Quiz

Tutorial: Contributor Training Series: Meshery UI (Amit Amrutiya)

Course: Contributing to Meshery CLI

Chapter: Building CLI

Section: <section>

Chapter: Documenting

Chapter: Designing commands Guidelines

Chapter: Unit Testing

Chapter: End-to-End Testing

Tutorial: Contributor Training Series: Meshery CLI (Aadhitya Amarendiran and Matthieu Evrin)

Course: End-to-End Testing in Meshery UI using Playwright

Chapter: <Chapter>

Section: <section>

Tutorial: Contributor Training Series: E2E Testing in Meshery UI (Ian Whitney)

Course: End-to-End Testing in Meshery CLI using BATS

Chapter: Introduction

Section: About Mesheryctl

Chapter: Setup Local Development Environment

Section: Prerequisites

Meshery CLI + Meshery Server installation
Provider account (e.g., Layer5 Cloud)
Kubernetes cluster (optional for k8s-related tests)
Tools: bash, jq, yq for processing JSON and YAML inputs

Section: Setup BATS core

MacOS (homebrew)
Any OS (npm)
Windows (from source via bash)

Section: Setup Dependencies

Section: Starting Meshery Server

Running Meshery for E2E tests
Using `make server`
When adapters/K8s are needed

Section: Authentication

Chapter: Folder Structure and Naming Conventions

Section: Understanding Meshery Repo Directories

`meshery/meshery` repo overview
Directories: `/mesheryctl`, `/server`, `/tests/e2e`

Section: E2E Test Folder Structure

Walkthrough of `mesheryctl/tests/e2e`
Explanation of helpers, setup/teardown scripts

Section: Test Naming Conventions

- Folder prefix (e.g., `002-model/`)
- File prefix (e.g., `01-model-list.bats`)
- Consistency rules

Chapter: Run End-to-End Tests Locally

Make sure you are in the meshery/mesheryctl directory

Section: Running All tests

`make e2e` (with build)

`make e2e-no-build` (without build)

Section: Running Specific Command Test Suite

```
make e2e-no-build BATS_FOLDER_PATTERN=<test folder name>
```

Section: Running Specific Test Files

```
make e2e-no-build BATS_FILE_PATTERN=<test folder name>
BATS_FILE_PATTERN=<test command name>
```

Section: Alternative Test Execution

- Run tests with already built binary

  Using `bash run_tests_local.sh`

- `Enforce rebuilding the binary`
  Forcing rebuild with `-b` flag

Chapter: Finding Issues to Work on

Section: Navigating Open Issues

Epic issue: [meshery/meshery#14031]

Create a sub-issue or comment down for the command you want to write tests for, or get yourself assigned a task under this area.

Track other tests progress, feel free to add reviews, and also take inspiration from how others have written the tests

Section: Using the Meshery Test Plan

Accessing the Meshery Test Plan (Sheet Views)

Chapter: Writing Tests with BATs

Official documentation is available at https://bats-core.readthedocs.io/en/stable/

The GitHub organization https://github.com/bats-core contains the bats-core repository and also the bats libraries repositories

Section: BATs Basics and interacting with mesheryctl

`@test` blocks
Run command execution
Capturing status and output

Section: Assertions

`assert_success`, `assert_failure`
`assert_output`, `assert_output --partial`
`assert_equal`, file assertions

Section: Setup and Teardown

Using `setup()` and `teardown()` to manage test environment

Chapter: Developing Your Tests

Chapter: Best Practices for Quality and Coverage

Section: Ensuring Quality

Use existing BATS utilities, avoid reinventing scripts
Keep test files clean, externalize data in fixtures
Add test scenarios in your PR, so that they can be added to the Meshery Test Plan

Section: Ensuring Coverage

Test core functionality
Test command flags
Test covering errors thrown by the command and invalid input files

Chapter: Reporting Bugs during Test Development

Section: Identifying bugs while writing tests

Section: Using the “mesheryctl Bug Report” template

Section: Providing Detailed Reports

Steps to reproduce
Expected vs actual behavior
Logs and environment details

Section: Linking Back to Tests

Mentioning the test case and file in the issue

Tutorial: Contributor Training Series: E2E Testing in Meshery CLI (Riya Garg)

Course: Build and Release

Chapter: <Chapter>

Section: <section>

Tutorial: Contributor Training Series: Meshery CI (Sangram Rath)

Course: Meshery Server

Chapter: <Chapter>

Section: <section>

Tutorial: Contributor Training Series: Meshery Server (Shlok Mishra)

Course: Meshery Models

Chapter: Models

Section: What are Meshery Models?

What exactly is a model? What can a model represent? Why are Models important in Meshery? How do Models help me manage apps, services, and infrastructure?

Section: Core Constructs of Model

Components, relationships, policies, connections and credentials, designs, patterns, metadata

Section: What is Model Schema

What's a schema, and why do we need it?
How is a schema different from the actual Model?

Section: Portability, Registry, and Intellectual Property

Why package models as OCI images?, What is a Registry and a registrant?

Section: Design Principles behind Meshery Models

Section: Entity Lifecycle in Meshery

Breaking down the confusing terms
Schema, Definition, Declaration, Instance

Section: Capabilities in Model

What are capabilities?

Section: Importing model

Section: Creating a model

Section: Post-Generation Enrichment

Section: How models are versioned

Tutorial: Contributor Training Series: Meshery Models (Aabid Sofi)

Labs:

Chapter: Components

Section: What Are Components?

Section: Semantics vs Non-Semantics Components

Section: Component Properties

Section: How to Contribute New Components

Prework:

1. Understand Model Generation and Packaging
    - Components exist within Models
    - Read through [Contributing to Models] first; without a model, a component is homeless
2. Customize Component Metadata & Representation
    - Form-based Representation

- Visual Representation

Development

3. Create Component Definition as a JSON file
4. Component Authoring Best Practices and Considerations
5. Contributing your component to the Meshery Project

Chapter: Relationships

Section: Introduction to Relationships

Section: Types of Relationships

Section: The Mechanics of Relationships

Anatomy of Relationship, Selectors, Actions, Operators

Section: How to create new Relationships

Prework - Relationship Identification, Relationship Classification
Development - Create a Relation definition as a JSON file, Configuring scope
Postwork - Testing and contribution

Course: Meshery Schema

Chapter: <Chapter>

Section: <section>

Course: Meshery Policies

Chapter: <Chapter>

Section: <section>

Course: Meshery Adapters

Chapter:

Certification