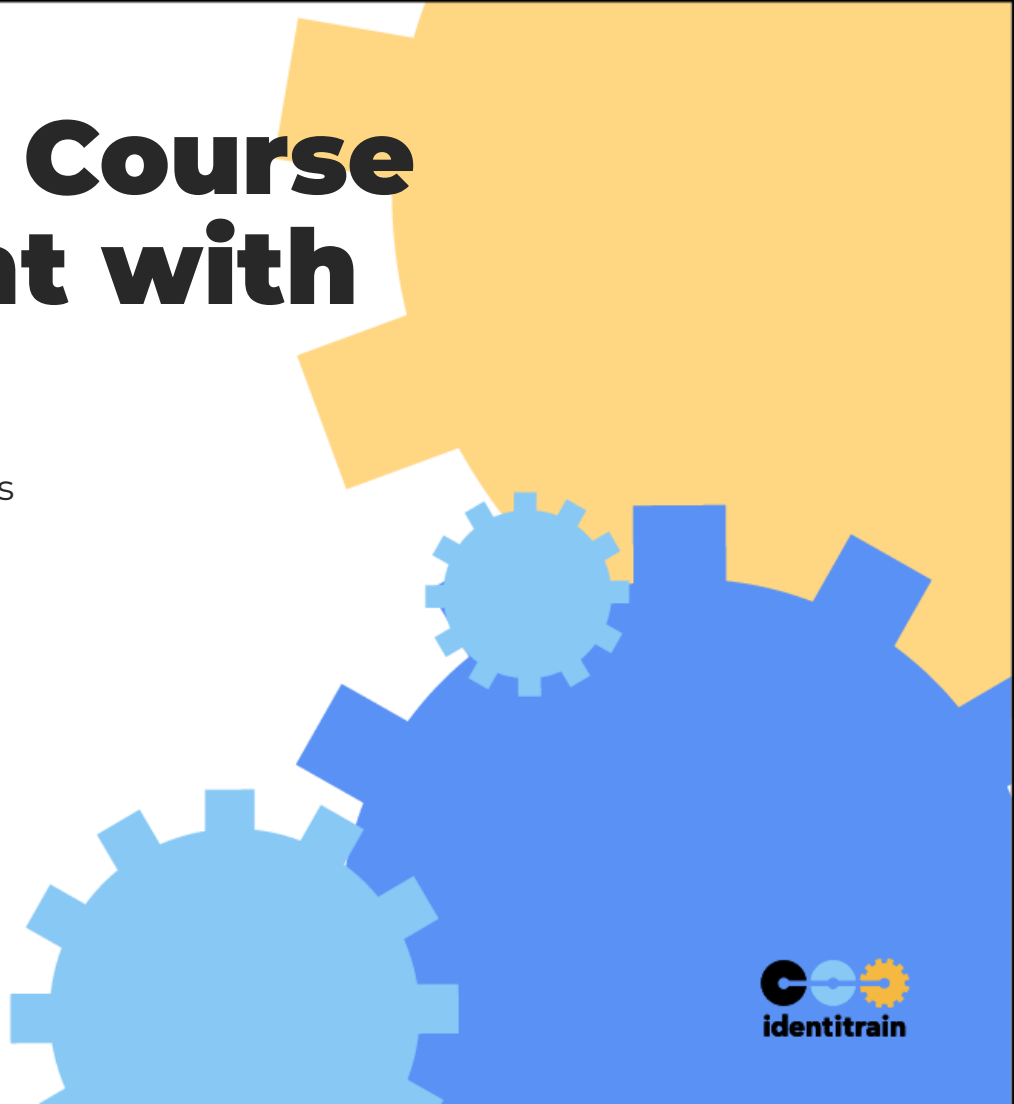
IDT-CD-001 - Course Development with CourseOps

Create Professional Training Materials with Markdown

identitrain

Welcome to Course Development with CourseOps

Building Professional Training Materials



identitrain

Course Goals

By the end of this course, you will be able to:

- **Set up your development environment** with VSCode and essential extensions
- **Understand the CourseOps architecture** and build pipeline
- **Create your first course** from scratch following best practices
- **Author content** using Markdown, images, and Mermaid diagrams
- **Publish courses** using Git workflow and automated builds
- **Generate multiple outputs** including HTML slides, PDFs, and EPUBs

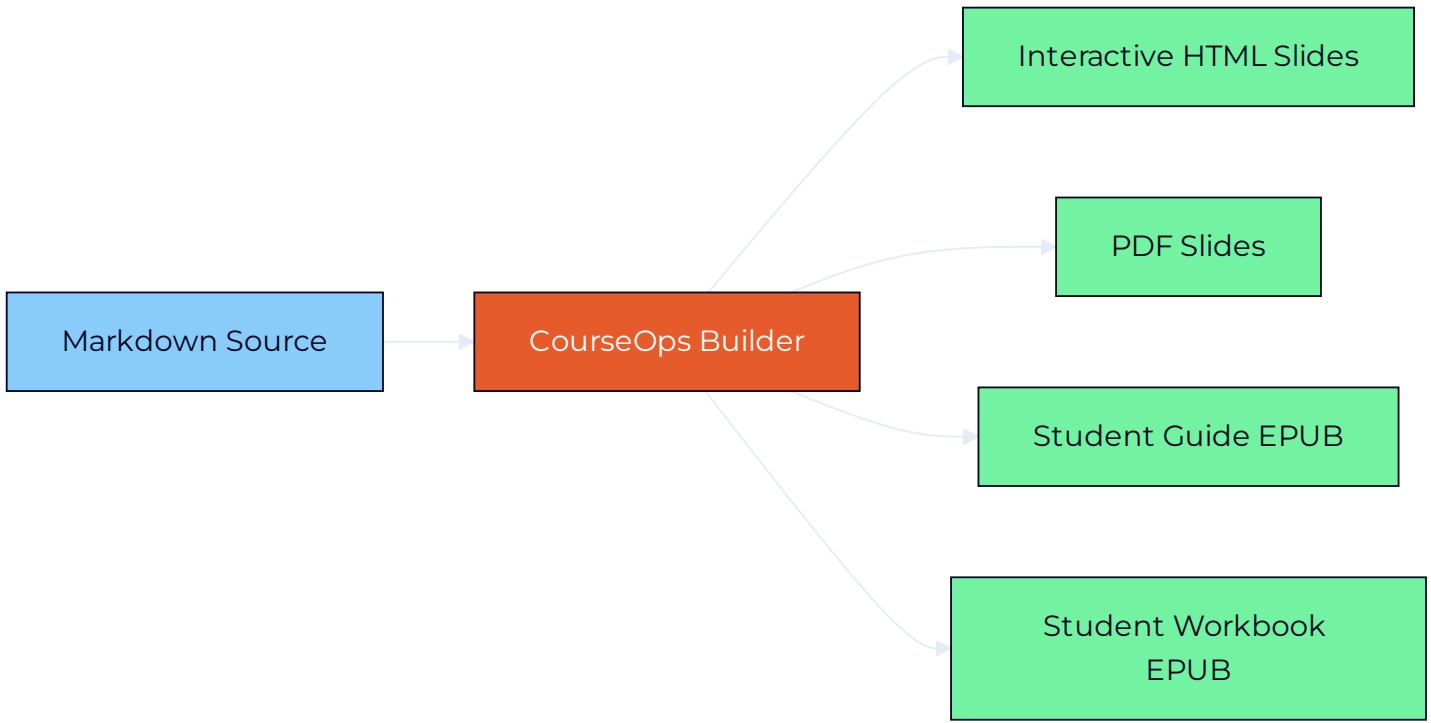
What is CourseOps?

CourseOps is an automated build and deployment system for training content

- **Write Once, Output Many** - Author in Markdown, get multiple formats
- **Version Control** - All courses stored in Git with full history
- **Automated Builds** - Push changes, get updated materials automatically
- **Professional Styling** - Consistent Identitrain branding
- **Easy Updates** - Fix typos or add content, rebuild in minutes

Output Formats

CourseOps automatically generates four output formats:



Why Use CourseOps?

Traditional Approach Problems

- □ Maintaining multiple file formats manually
- □ Inconsistent styling across materials
- □ Time-consuming updates to all formats
- □ No version control for content changes
- □ Difficult collaboration between authors

CourseOps Solutions

- □ Single source of truth in Markdown
- □ Automated, consistent styling
- □ Update once, rebuild in minutes
- □ Full Git version control
- □ Easy collaboration and review

Course Prerequisites

To succeed in this course, you should have:

- **Basic Markdown** - Familiarity with Markdown syntax
  - Headers, lists, links, images
  - Code blocks and formatting
- **Git Fundamentals** - Understanding of version control
  - Clone, commit, push, pull
  - Basic branching concepts
- **Text Editor Experience** - Comfortable editing text files
  - VSCode preferred (we'll set it up)
  - Any programmer's text editor works

identitrain

Course Structure

Module Overview

1. **Development Environment** - Setting up VSCode and extensions
2. **Architecture** - Understanding the CourseOps system
3. **Creating Your First Course** - Hands-on course creation
4. **Working with Content** - Markdown, images, and diagrams
5. **Publishing** - Git workflow and automated builds
6. **Wrap-up** - Best practices and next steps

Let’s Get Started!

What’s Next?

- Setting up your development environment
- Installing VSCode and essential extensions
- Configuring your workspace for maximum productivity

Ready to begin? Let’s set up your development environment!

Development Environment Setup

Configuring VSCode for Course Development

identitrain

Why Visual Studio Code?

VSCode is the recommended editor for course development

- **Free and Open Source** - No licensing costs
- **Cross-Platform** - Windows, macOS, Linux
- **Excellent Markdown Support** - Real-time preview, syntax highlighting
- **Git Integration** - Built-in version control
- **Rich Extension Ecosystem** - Thousands of helpful plugins
- **Active Development** - Regular updates and improvements

Download: https://code.visualstudio.com/

identitrain

Installing VSCode Extensions
Three Installation Methods

Method 1: Extensions View (Recommended)

1. Open VSCode
2. Click Extensions icon (left sidebar)
3. Search for extension name
4. Click **Install**

Method 2: Extension URLs

- Click marketplace links (we'll provide these)
- Browser opens → Click **Install**
- Confirm in VSCode

Method 3: Command Line

```
code --install-extension publisher.extension-id
```

identitrain

Essential Extension #1: Markdown Preview Enhanced

By: Yiyi Wang (formerly Haixin Chen) **Purpose:** Live preview of Markdown with Mermaid support

Markdown Preview Enhanced Features

Key capabilities for course development:

- **Real-time Rendering** - See changes as you type
- **Mermaid Diagrams** - Visualize flowcharts, sequences, etc.
- **Math Equations** - LaTeX support for formulas
- **Code Highlighting** - Syntax highlighting for code blocks
- **Export Options** - PDF, HTML, EPUB export
- **Presentation Mode** - Present directly from preview

Install:

```
code --install-extension shd101wyy.markdown-preview-enhanced
```

Usage: Open any `.md` file → Press `Ctrl+K V` (Win/Linux) or `Cmd+K V` (Mac)

identitrain

Essential Extension #2: Slidev

By: Anthony Fu **Purpose:** Official Slidev extension with enhanced slide authoring features

Slidev Extension Features

Enhanced support for slide development:

- **Slidev Syntax Highlighting** - Proper colors for frontmatter and directives
- **Slide Folding** - Collapse/expand individual slides
- **Frontmatter Validation** - Catch configuration errors early
- **Quick Navigation** - Jump between slides easily
- **Code Snippets** - Common Slidev patterns
- **Hover Documentation** - Inline help for configuration options

Install:

```
code --install-extension antfu.slidev
```

identitrain

Essential Extension #3: YAML

By: Red Hat **Purpose:** Validation and auto-completion for YAML frontmatter

YAML Extension Features

Essential for configuration management:

- **Syntax Validation** - Catch YAML errors immediately
- **Auto-completion** - Suggests valid keys and values
- **Hover Documentation** - See what each field does
- **Format on Save** - Keep configuration clean
- **Schema Validation** - Verify against known patterns
- **Error Highlighting** - Red squiggles for problems

Install:

```
code --install-extension redhat.vscode-yaml
```

identitrain

Essential Extension #4: Mermaid Syntax Highlighting

By: Brian Pruitt-Goddard **Purpose:** Syntax highlighting for Mermaid diagrams in Markdown

Mermaid Syntax Highlighting Features

Makes diagram code readable:

- **Color-coded Syntax** - Distinguish nodes, edges, labels
- **Node Type Highlighting** - Visual distinction between elements
- **Error Detection** - Spot syntax problems early
- **Improved Readability** - Complex diagrams easier to understand

Install:

```
code --install-extension bpruitt-goddard.mermaid-markdown-syntax-highlighting
```

Works automatically in all ```` ```mermaid ```` code blocks

identitrain

Essential Extension #5: Claude Code

By: Anthropic **Purpose:** AI-powered coding assistant for content development

Claude Code Features

AI assistance while you work:

- **Natural Language Editing** - Ask for changes in plain English
- **Content Suggestions** - Get ideas for explanations
- **Markdown Formatting** - Help with syntax and structure
- **Code Block Generation** - Create examples in any language
- **Grammar and Style** - Writing improvement suggestions
- **Context-Aware** - Understands your entire file

Install:

```
code --install-extension anthropic.claude-code
```

Usage: Press `Ctrl+L` / `Cmd+L` to open Claude Code

Additional Recommended Extensions

Git Management

Git Graph by mhutchie

- Visualize repository history
- Interactive commit graph
- Branch management

GitLens by GitKraken

- Line-by-line blame annotations
- File and line history
- Commit search and comparison

Install:

```
code --install-extension mhutchie.git-graph
code --install-extension eamodio.gitlens
```

Git Graph Extension

GitLens Extension

Code Quality Extension
Code Spell Checker

By: Street Side Software **Purpose:** Catch spelling mistakes in your course content

Install All Extensions at Once

Copy and paste this command to install all essential extensions:

```
# Essential extensions
code --install-extension shd101wyy.markdown-preview-enhanced
code --install-extension antfu.slidev
code --install-extension redhat.vscode-yaml
code --install-extension bpruitt-goddard.mermaid-markdown-syntax-highlighting
code --install-extension anthropic.claude-code

# Additional recommended
code --install-extension mhutchie.git-graph
code --install-extension eamodio.gitlens
code --install-extension streetsidesoftware.code-spell-checker
```

Note: Restart VSCode after installing all extensions

identitrain

VSCode Configuration
Recommended Settings

Create `.vscode/settings.json` in your course repository:

```
{
  "markdown.preview.breaks": true,
  "editor.wordWrap": "on",
  "editor.fontSize": 14,
  "editor.lineHeight": 1.6,
  "files.associations": {
    "*.md": "markdown"
  },
  "git.enableSmartCommit": true,
  "git.confirmSync": false
}
```

identitrain

Essential Keyboard Shortcuts

Master these shortcuts for maximum productivity:

Action	Windows/Linux	macOS
Toggle Preview	`Ctrl+K V`	`Cmd+K V`
Bold Text	`Ctrl+B`	`Cmd+B`
Italic Text	`Ctrl+I`	`Cmd+I`
Command Palette	`Ctrl+Shift+P`	`Cmd+Shift+P`
Quick File Open	`Ctrl+P`	`Cmd+P`
Find in Files	`Ctrl+Shift+F`	`Cmd+Shift+F`
Open Claude Code	`Ctrl+L`	`Cmd+L`

identitrain

Environment Setup Complete!

What We've Accomplished

- ☐ Installed Visual Studio Code
- ☐ Configured essential extensions
- ☐ Set up workspace preferences
- ☐ Learned key shortcuts

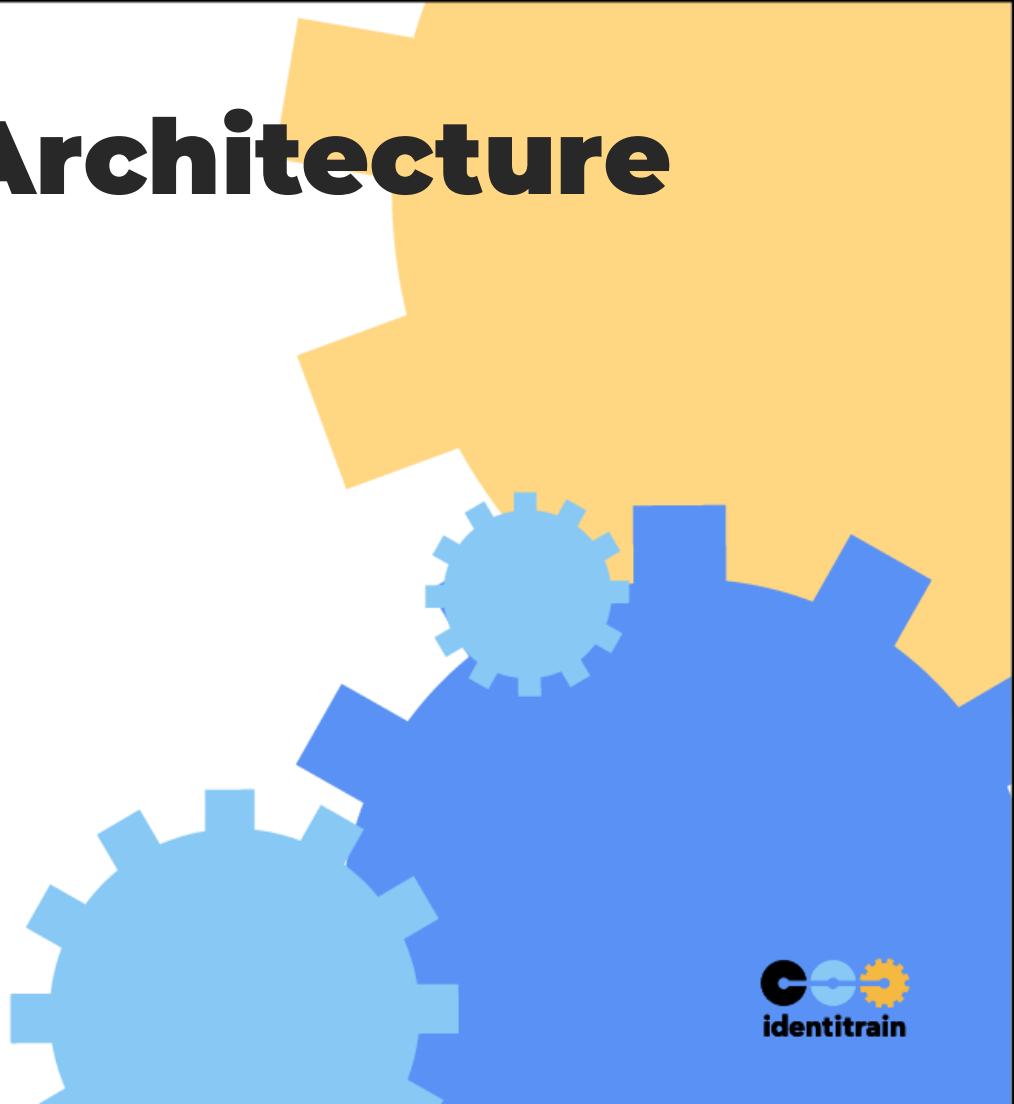
What's Next?

Understanding the CourseOps architecture and build pipeline

identitrain

CourseOps Architecture

Understanding the Build System



identitrain

Multi-Repository Architecture

CourseOps uses separate Git repositories for each component:

Repository Roles

idt-courseops-catalog

- **Central Registry** - Lists all available courses
- **Contains** - `courses.yaml` configuration file
- **Defines** - Which courses to build and from where

IDT-XXX-NNN (Course Repositories)

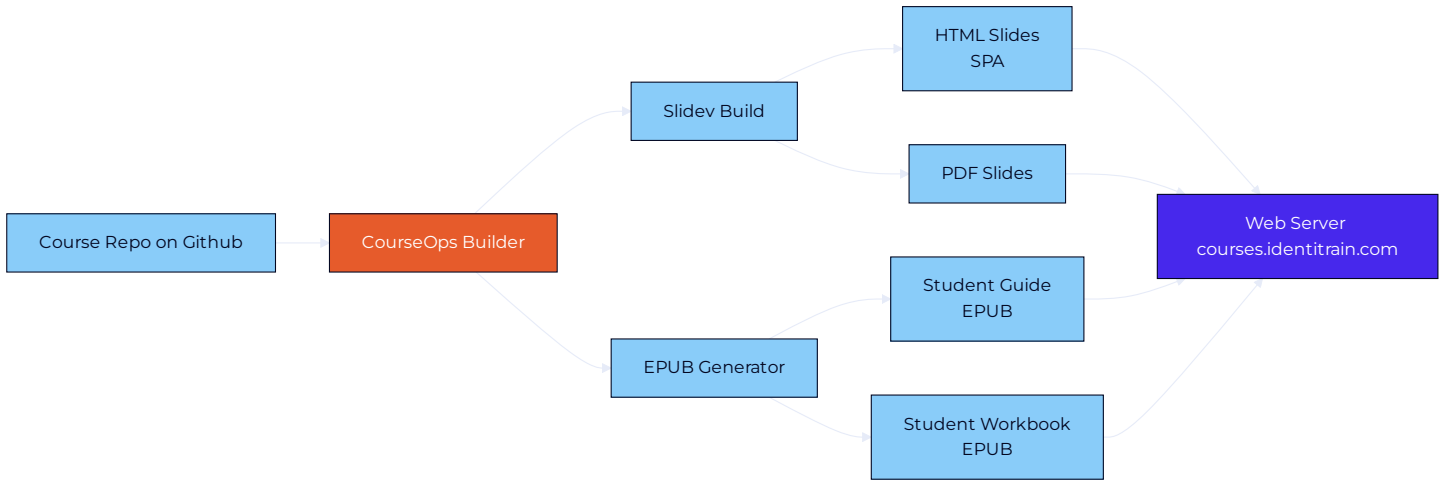
- **Course Content** - All Markdown, images, and assets
- **Independent** - Each course is self-contained
- **Versioned** - Full Git history per course

courseops

- **Build System** - Python and Node.js build scripts
- **Infrastructure** - Docker containers and K8s configs
- **Rarely Modified** - Stable build platform

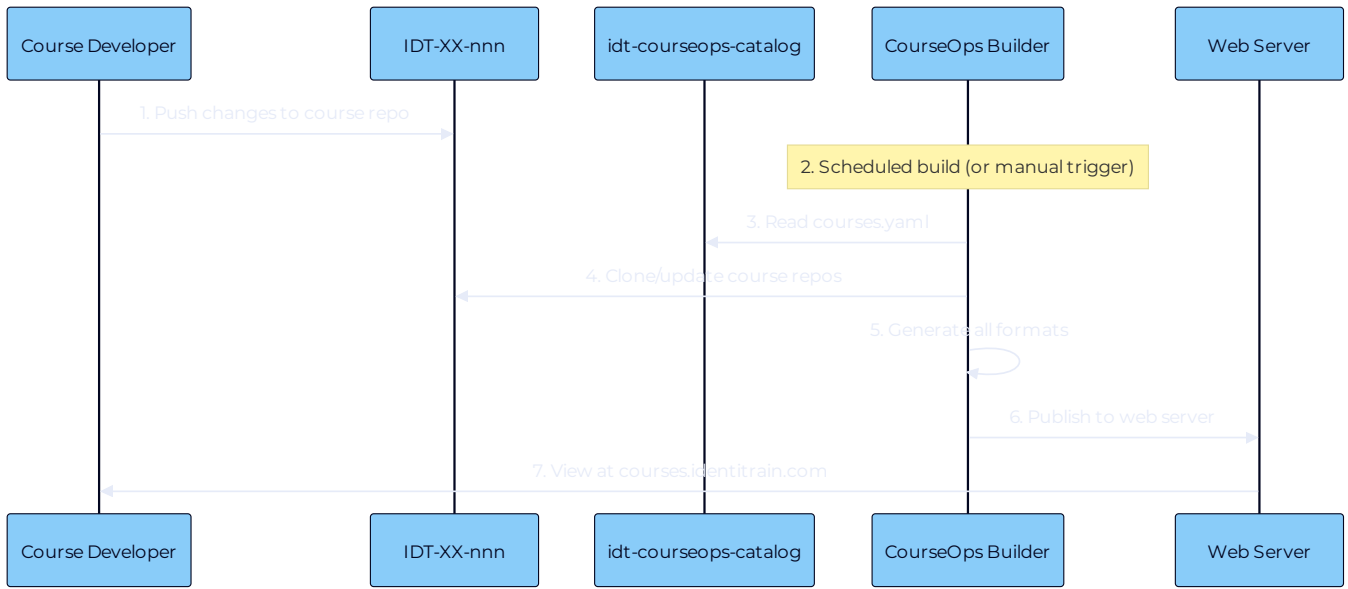
The Build Pipeline

How your Markdown becomes training materials:



Build Trigger Flow

How builds are initiated and executed:



The courses.yaml File

The catalog defines which courses to build:

```yaml
catalog_version: 1
courses:
  - slug: idt-ai-101
    repo_ssh_url: git@github.com:identitrain/IDT-AI-101.git
    ref: main
  - slug: idt-cd-001
    repo_ssh_url: git@github.com:identitrain/IDT-CD-001.git
    ref: main
```

Key Fields:

- **slug** - Lowercase course identifier (used in URLs)
- **repo_ssh_url** - SSH URL to course repository
- **ref** - Branch or tag to build (usually `main`)

identitrain

State Management

CourseOps tracks what's been built:

- **State File** - `.courseops-build-state.json`
- **Tracks** - Git commit SHA for each course
- **Smart Rebuilds** - Only rebuilds changed courses
- **Force Rebuild** - Option to rebuild everything

How it works:

1. Builder checks state file for last built SHA
2. Compares with current repository SHA
3. Skips build if SHAs match
4. Rebuilds and updates state if different

identitrain

Course Repository Structure

A standard course repository layout:

```
IDT-XX-NNN/                    # Your course code
├── slides.md                   # Main file (required)
├── 0-intro.md                  # Introduction chapter
├── 1-ChapterName.md           # Chapter files
├── 2-AnotherChapter.md
├── 4-end.md                   # Conclusion
├── images/                     # All image assets
│   ├── diagram1.png
│   └── screenshot.png
├── exercises/                  # Exercise files
│   └── Chapter 1.md
├── .gitignore                  # Git ignore rules
└── README.md                   # Course documentation
```

The slides.md File

Chapter References

Output Directory Structure

What gets generated and where it goes:

```
courses.identitrain.com/
├── index.html                      # Course catalog
├── _courseops_theme/               # Shared assets
│   ├── index.css
│   ├── fonts/
│   └── assets/
└── idt-cd-001/                     # Your course
    ├── index.html                  # Interactive slides
    ├── IDT-CD-001_revA_Slides.pdf
    ├── IDT-CD-001_revA_Guide.epub
    └── IDT-CD-001_revA_Workbook.epub
```

Local vs. Cloud Builds

Cloud Builds (Production)

- **Automatic** - Runs on schedule or manual trigger
- **Infrastructure** - Kubernetes on Google Cloud
- **Output** - Published to courses.identitrain.com
- **Full Pipeline** - All courses, all formats

Local Builds (Development)

- **On-Demand** - Run when you need to test
- **Docker-based** - Runs in local container
- **Output** - Local `_local_out/` directory
- **Single Course** - Test your changes quickly

Docker Build Environment

The builder runs in a containerized environment:

- **Base Image** - Ubuntu with Node.js and Python
- **Slidev** - Installed globally for slide generation
- **Puppeteer** - For PDF export from HTML
- **Python Dependencies** - PyYAML for parsing configs
- **Git** - For cloning course repositories

Benefits:

- **Consistency** - Same environment everywhere
- **Isolation** - Builds don't interfere with each other
- **Reproducibility** - Same inputs → same outputs

identitrain

Build Performance

The builder is optimized for speed:

- **Parallel Builds** - Multiple courses build simultaneously
- **State Tracking** - Skip unchanged courses
- **Shallow Clones** - Fetch only needed Git history
- **Incremental Outputs** - Update only changed files
- **Cached Dependencies** - Reuse installed packages

Typical Build Times:

- Single unchanged course: ~5 seconds (skipped)
- Single changed course: ~2-5 minutes
- Full catalog rebuild: ~15-30 minutes

identitrain

Kubernetes Deployment

The build system runs on Kubernetes (GKE):

Components:

- **CronJob** - Scheduled builds (daily)
- **PersistentVolume** - Stores work directories and state
- **Nginx Deployment** - Serves course outputs
- **Ingress** - Routes traffic to courses.identitrain.com

Benefits:

- **Scalability** - Handle any number of courses
- **Reliability** - Automatic restarts on failure
- **Resource Management** - CPU/memory limits
- **Logging** - Centralized build logs

identitrain

Architecture Summary

Key Concepts

- **Multi-Repository** - Each course is independent
- **Catalog-Driven** - courses.yaml controls what builds
- **Automated Pipeline** - Markdown → Multiple formats
- **State Management** - Smart rebuilds save time
- **Containerized** - Consistent builds everywhere
- **Cloud-Native** - Kubernetes for production

What's Next?

Creating your first course from scratch!

identitrain

Creating Your First Course

From Concept to Published Material

identitrain

Step 1: Plan Your Course

Before creating files, plan your course structure:

Choose a Course Code

Format: `IDT-XX-NNN`

- `IDT` = Identitrain prefix
- `XX` = 2-letter subject code
- `NNN` = 3-digit number

Examples:

- `IDT-AI-101` - AI for Identity Management
- `IDT-GD-200` - Groovy Development
- `IDT-CD-001` - Course Development

Define Course Metadata

Plan these details before you start:

- **Title** - Full course name
  - Example: "Course Development with CourseOps"
- **Duration** - Length in days
  - Example: 1 day, 3 days, 5 days
- **Target Audience** - Who should attend
  - Example: "Course Developers, Technical Writers"
- **Prerequisites** - Required knowledge
  - Example: "Basic Markdown, Git fundamentals"
- **Description** - Course overview (2-4 sentences)

identitrain

Outline Content Structure

Break your course into logical chapters:

Typical Structure:

1. **Introduction** (0-intro.md)
   - Welcome and objectives
   - Prerequisites review
   - Course overview
2. **Main Chapters** (1-xxx.md, 2-xxx.md, etc.)
   - 3-8 chapters recommended
   - Each chapter: 15-45 minutes
   - Include practical examples
3. **Conclusion** (N-end.md)
   - Summary of key points
   - Next steps

Step 2: Create Directory Structure

Create the course directory and subdirectories:

```
# Create main course directory
mkdir IDT-XX-NNN
cd IDT-XX-NNN

# Create subdirectories
mkdir images

# Create initial files
touch slides.md README.md .gitignore
touch 0-intro.md 4-end.md
```

Result:

```
IDT-XX-NNN/
├── slides.md
├── README.md
├── .gitignore
├── 0-intro.md
├── 4-end.md
└── images/
```

Create .gitignore File

Prevent build artifacts from being committed:

```
cat > .gitignore << 'EOF'
# Build outputs
*.pdf
*.epub
dist/
.slidev/

# OS files
.DS_Store
Thumbs.db

# Editor files
.vscode/
.idea/
*.swp

# Node modules
node_modules/

# Temporary files
*.tmp
```

Step 3: Initialize Git Repository

Set up version control:

```
# Initialize git
git init

# Stage all files
git add .

# Make initial commit
git commit -m "Initial course structure for IDT-XX-NNN"

# Create main branch
git branch -M main
```

What this does:

- Creates Git repository
- Stages initial files
- Makes first commit
- Renames default branch to "main"

Step 4: Create GitHub Repository On GitHub:

1. Navigate to `https://github.com/identitrain`
2. Click **"New Repository"**
3. Name: `IDT-XX-NNN` (exact course code, uppercase)
4. Description: Your course title
5. Visibility: Private (or per policy)
6. **DO NOT** initialize with README
7. Click **"Create repository"**

identitrain

Connect Local to GitHub

Push your local repository to GitHub:

```
# Add GitHub remote
git remote add origin git@github.com:identitrain/IDT-XX-NNN.git

# Push to main branch
git push -u origin main
```

What this does:

- Links local repository to GitHub
- Pushes your initial commit
- Sets up tracking between local and remote

Verify: Visit your GitHub repository to see your files

identitrain

Step 5: Create slides.md

The main course file with frontmatter:

```
---
# Slidev configuration
theme: identitrain
highlighter: shiki
lineNumbers: false
drawings:
  persist: false
transition: slide-left
download: true
favicon: favicon.ico
selectable: true

# Course metadata
title: Your Complete Course Title
exportFilename: IDT-XX-NNN_revA_Slides
idtCourseCode: IDT-XX-NNN
idtCourseRevision: A
idtCourseDuration: 1
...
```

identitrain

Frontmatter Metadata Fields

Add Chapter References

Below the frontmatter, reference your chapter files:

```
# IDT-XX-NNN - Your Course Title

Brief subtitle or tagline

---
src: ./0-intro.md
hidden: false
---

---
src: ./1-FirstChapter.md
hidden: false
---

---
src: ./2-SecondChapter.md
hidden: false
---

---
src: ./4-end.md
```

Course Structure in VSCode

Step 6: Write Your First Chapter

Create `0-intro.md` with this structure:

```
---
layout: cover
---

# Chapter Title

Subtitle or brief description

<!--
Instructor notes for this slide

=== NOTES ===

Detailed notes that appear in student guides
-->

---

# Content Slide

* Bullet point one
* Bullet point two
```

Step 7: Commit Your Changes

Save your work with Git:

```
# Check status
git status

# Stage changes
git add .

# Commit with message
git commit -m "Add course frontmatter and introduction chapter"

# Push to GitHub
git push origin main
```

Best Practices:

- Commit frequently (after each significant change)
- Write clear commit messages
- Push regularly to back up work

Step 8: Add to Catalog

Edit `idt-courseops-catalog/courses.yaml`:

```yaml
catalog_version: 1
courses:
  - slug: idt-ai-101
    repo_ssh_url: git@github.com:identitrain/IDT-AI-101.git
    ref: main
  # Add your course here
  - slug: idt-xx-nnn
    repo_ssh_url: git@github.com:identitrain/IDT-XX-NNN.git
    ref: main
```

Important:

- `slug` must be lowercase
- `repo_ssh_url` must be SSH format (not HTTPS)
- `ref` is typically `main`

identitrain

Catalog Repository Workflow

```
# Clone catalog (first time only)
git clone git@github.com:identitrain/idt-courseops-catalog.git

# Navigate to catalog
cd idt-courseops-catalog

# Edit courses.yaml
# (add your course entry)

# Commit and push
git add courses.yaml
git commit -m "Add IDT-XX-NNN to catalog"
git push origin main
```

identitrain

Step 9: Trigger a Build

Automatic Builds

Builds typically run on a schedule (daily)

Manual Trigger (for testing)

1. Go to Google Cloud Console
2. Navigate to Kubernetes → Workloads
3. Find `courseops-builder` CronJob
4. Click **"Run now"**

Wait: Builds take 5-15 minutes depending on course size

Step 10: Verify Your Course

Once built, check your course at:

Slides: `https://courses.identitrain.com/idt-xx-nnn/`

Verify:

- ☐ Slides render correctly
- ☐ Navigation works
- ☐ Images display properly
- ☐ Mermaid diagrams render
- ☐ PDF downloads correctly
- ☐ EPUBs open and contain content

Common First-Course Issues

Images Don’t Display

- Check file path: `images/filename.png`
- Verify file is committed to Git
- Ensure filename case matches

Build Fails

- Check `courses.yaml` syntax (valid YAML)
- Verify SSH URL is correct
- Ensure `slug` is lowercase

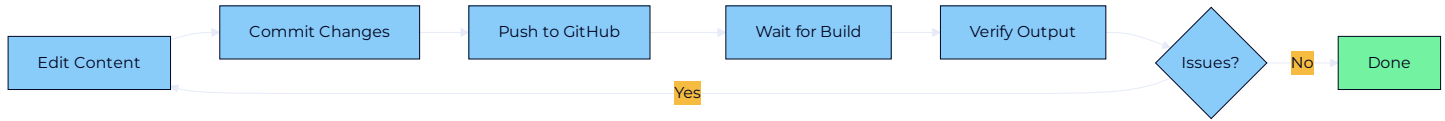
Mermaid Doesn’t Render

- Check code block syntax: ```` ```mermaid ````
- Validate diagram at mermaid.live
- Ensure no indentation before block

identitrain

Iteration Workflow

Once your course is set up, the typical workflow is:



Typical Cycle:

1. Edit content locally in VSCode
2. Commit changes with Git
3. Push to GitHub
4. Trigger build (or wait for scheduled)
5. Verify output
6. Repeat until satisfied

First Course Complete!

What We've Accomplished

- ✅ Planned course structure and metadata
- ✅ Created directory structure
- ✅ Initialized Git repository
- ✅ Connected to GitHub
- ✅ Created slides.md with frontmatter
- ✅ Wrote first chapter
- ✅ Added course to catalog
- ✅ Built and verified output

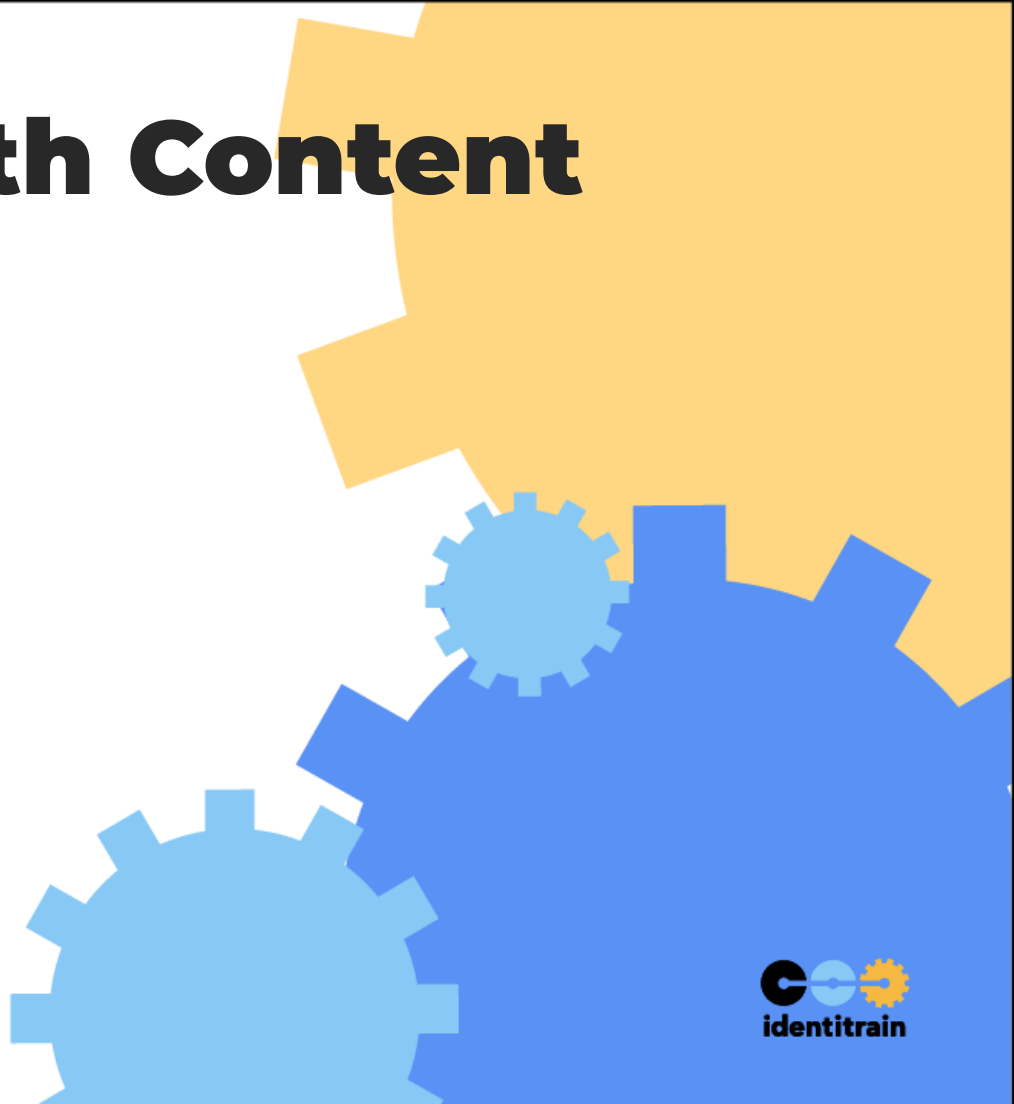
What's Next?

Working with content: Markdown, images, and Mermaid diagrams

identitrain

Working with Content

Markdown, Images, and Diagrams



Markdown Basics

Markdown is a lightweight markup language for formatting text:

```
# Headers
## Level 2
### Level 3

**bold text**
*italic text*
`inline code`

* Bullet list
* Another item

1. Numbered list
2. Second item

[Link text](https://example.com)
```

identitrain

Slide Layouts

Slidev supports multiple layouts for different slide types:

cover - Chapter titles and section breaks

```
---
layout: cover
---

# Chapter Title
Subtitle or description
```

default - Standard content slides

```
---
layout: default
---

# Slide Title
Content goes here
```

Two-Column Layout

Split content across two columns:

```
---
layout: two-cols
---

# Left Column Content

* Point one
* Point two
* Point three

::right::

# Right Column Content

* Different points
* On the right side
* Next to left content
```

Use cases:

- Before/After comparisons

Code Blocks

Display code with syntax highlighting:

````
<CodeBlockWrapper v-bind="{}" :title='""' :ranges='[]'>

```javascript
function greet(name) {
  console.log(`Hello, ${name}!`);
  return true;
}
```

</CodeBlockWrapper>


<CodeBlockWrapper v-bind="{}" :title='""' :ranges='[]'>

```python
def greet(name):
    print(f"Hello, {name}!")
    return True
```

</CodeBlockWrapper>
````

Inline Code and Emphasis

Format inline content:

- `Inline code` - Use backticks for code, commands, filenames
- **Bold text** - Use double asterisks for emphasis
- *Italic text* - Use single asterisks for subtle emphasis
- ***Bold italic*** - Combine for strong emphasis

Guidelines:

- Use `code formatting` for technical terms, filenames, commands
- Use **bold** for important concepts or warnings
- Use *italics* sparingly for subtle emphasis
- Don't overuse formatting - it loses impact

identitrain

Lists and Structure

Create organized, scannable content:

Unordered Lists:

```
* Main point
  * Sub-point with indentation
  * Another sub-point
* Second main point
```

Ordered Lists:

```
1. First step
2. Second step
   1. Sub-step
   2. Another sub-step
3. Third step
```

Tips:

- Keep lists concise (3-7 items ideal)
- Use parallel structure (all verbs or all nouns)

Links and References

Create clickable links:

```
[Link text](https://example.com)
[Link with title](https://example.com "Hover text")
```

Best Practices:

- Use descriptive text (not "click here")
- Verify links work before publishing
- Consider link longevity (no temporary URLs)
- For internal references, use relative paths

Examples:

- Good: VSCode Download
- Poor: Click here

identitrain

Working with Images

Images enhance understanding and engagement:

Basic Syntax:

```
![Alt text](./images/diagram.png)
```

Best Practices:

- Store all images in `images/` directory
- Use descriptive filenames: `auth-flow.png` not `img1.png`
- Include alt text for accessibility
- Optimize file sizes (keep under 500KB)

Supported Formats:

- PNG - Screenshots, diagrams with transparency
- JPG - Photographs
- SVG - Vector graphics (preferred for diagrams)

Image Sizing and Placement

Control image display:

Default:

```
![Description](./images/photo.png)
```

Custom Size:

```
<img src="images/large.png" style="width: 80%;" />
```

With Caption:

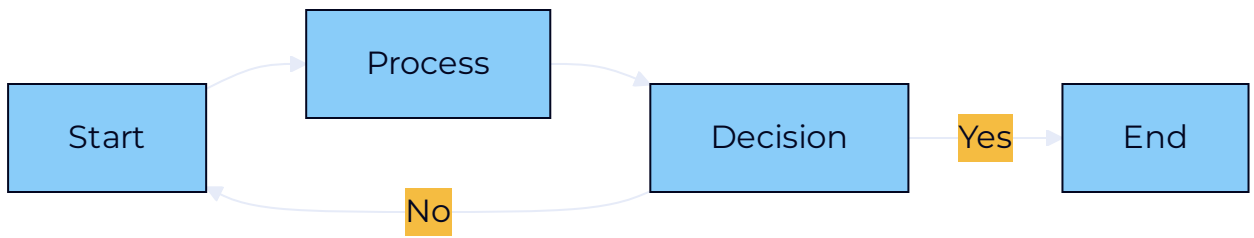
```
![Architecture diagram](./images/arch.png)
*Figure 1: System architecture overview*
```

Creating Mermaid Diagrams

Mermaid generates diagrams from text:

```
<Mermaid code-lz="OYJwhgDgFgBAMgJQFA1TAggbQMoBcwi4C6MAtKQHwwBCmACiAPYDGApgM7tEprVmUwAwpgAirZgEt2ExgDtu
```

Result:



Flowchart Diagrams

Model processes and workflows:

```
<Mermaid code-lz="GYGw9g7gxgFghgJwC4AIAqARAUC3KCCA2gKoDOApgigDJgDmAlgHYC6KAtOwHwoBCA3gDU4IBgBMA/AF8ceXl
```

Result:

Sequence Diagrams

Show interactions between components:

```
<Mermaid code-lz="M4UwjgriB2DGIBECWBDA5gJxQWwFAAJD8AHFDAFyViVOnPwFVQMCjSKqaU78BBY4q0LtK1WvV4AFAJJCSZUV
```

Class and ER Diagrams

Model data structures and relationships:

```
<Mermaid code-lz="MYGwhgzhAiCWYHMBOYC2AoABNzpIUwFUIBTJTAbyxxoGoBlAFyVgDsFMBXUpVtE6jWwNmbDiVRhYIQUNphOj
```

Mermaid Best Practices

Create clear, effective diagrams:

- **Keep it Simple** - Aim for 5-12 nodes maximum
- **Use Descriptive Labels** - Clear, concise text
- **Be Consistent** - Same style throughout course
- **Test First** - Validate at mermaid.live
- **Add Color Sparingly** - For emphasis only
- **Consider Complexity** - Split complex diagrams

Styling Example:

```
style NodeName fill:#4728ec,color:#fff
```

identitrain

Speaker Notes and Student Guides

Add two types of notes to slides:

```
---
# Slide Content

* Visible bullet points

<!--
Instructor notes: Delivery tips, timing, stories

==== NOTES ====

Detailed content for student guide EPUB.
This appears in the e-book but not on slides.
Use **Markdown** formatting here.
-->
```

Two Sections:

1. Before `==== NOTES ====` - Instructor guidance
2. After `==== NOTES ====` - Student guide content

Tables in Markdown

Create formatted tables:

```
| Column 1 | Column 2 | Column 3 |
|----------|----------|----------|
| Data 1   | Data 2   | Data 3   |
| Data 4   | Data 5   | Data 6   |
```

Alignment:

```
| Left | Center | Right |
|:-----|:------:|------:|
| L    |   C    |     R |
```

Tips:

- Keep tables simple (4-6 columns max)
- Use for structured data, not layout
- Consider lists for simple comparisons

identitrain

Block Quotes

Highlight important information:

```
> This is a quoted section
>
> Multiple paragraphs are supported
```

Use for:

- Important definitions
- Key takeaways
- Callouts and warnings

Example:

> **Important:** Always commit your changes before switching branches to avoid losing work.

identitrain

Content Organization Tips

Structure content for maximum learning:

- **One Concept Per Slide** - Don't overload
- **Progressive Disclosure** - Build complexity gradually
- **Visual Hierarchy** - Headers, bullets, emphasis
- **White Space** - Don't fill every pixel
- **Consistent Style** - Patterns aid comprehension

Example Flow:

1. Introduce concept (overview slide)
2. Explain details (breakdown slides)
3. Show example (code or diagram)
4. Reinforce learning (summary slide)

identitrain

Accessibility Considerations

Make content accessible to all learners:

- **Alt Text** - Describe images clearly
- **Color** - Don't rely solely on color
- **Contrast** - Ensure text is readable
- **Structure** - Use proper heading hierarchy
- **Clear Language** - Avoid unnecessary jargon

Example:

```
![Flowchart showing authentication process with four steps: user login, credential verification, token
```

Content Creation Summary

Key Techniques

- **Markdown** - Simple syntax, powerful formatting
- **Images** - Store in `images/`, optimize sizes
- **Mermaid** - Text-based diagrams that auto-generate
- **Layouts** - Cover, default, two-column for variety
- **Notes** - Instructor guidance + student guide content
- **Structure** - One concept per slide, clear hierarchy

What's Next?

Publishing your course with Git and automated builds

identitrain

Publishing Your Course

Git Workflow and Automated Builds

identitrain

The Publishing Workflow

From local changes to published course:

Daily Git Workflow

Check Status

```
git status
```

Shows:

- Modified files (red)
- Staged files (green)
- Untracked files
- Current branch

Staging Changes

Stage All Changes

```
git add .
```

Stage Specific Files

```
git add slides.md
git add 1-Chapter.md
git add images/new-diagram.png
```

Stage by Pattern

```
git add *.md          # All Markdown files
git add images/*.png  # All PNG images
```

identitrain

Creating Commits

Basic Commit

```
git commit -m "Add chapter 2 on authentication"
```

Commit with Description

```
git commit -m "Add chapter 2 on authentication" -m "Covers password auth, MFA, and SSO. Includes examp
```

Best Practices:

- Write clear, descriptive messages
- Use present tense: "Add" not "Added"
- Explain *why*, not just *what*
- Keep first line under 50 characters

identitrain

Pushing to GitHub

Push to Remote

```
git push origin main
```

First Push (with upstream tracking)

```
git push -u origin main
```

What Happens:

- Local commits upload to GitHub
- GitHub repository updates
- Build system can now access changes
- Collaborators can see updates

Viewing History

See Recent Commits

```
git log
git log --oneline
git log --graph --oneline --all
```

See Changes in Commit

```
git show <commit-hash>
```

See What Changed

```
git diff                    # Unstaged changes
git diff --staged           # Staged changes
git diff HEAD~1 HEAD        # Last commit vs current
```

identitrain

Branching Basics

Create and Switch to Branch

```
git checkout -b feature/new-chapter
```

Switch Between Branches

```
git checkout main
git checkout feature/new-chapter
```

List Branches

```
git branch              # Local branches
git branch -a           # All branches
```

Merging Changes

Merge Branch to Main

```
# Switch to main
git checkout main

# Merge feature branch
git merge feature/new-chapter

# Push updated main
git push origin main
```

Delete Merged Branch

```
git branch -d feature/new-chapter
```

Handling Merge Conflicts

When the same lines were changed in both branches:

```
# Git tells you there are conflicts
Auto-merging 1-Chapter.md
CONFLICT (content): Merge conflict in 1-Chapter.md
```

To Resolve:

1. Open conflicted file in VSCode
2. Look for conflict markers:

   ```
   <<<<<<< HEAD
   Your version
   =======
   Other version
   >>>>>>> branch-name
   ```

3. Edit to keep desired content
4. Remove conflict markers
5. Stage and commit resolved file

Pull Before Push

Always sync before pushing:

```
# Pull latest changes from GitHub
git pull origin main

# If conflicts, resolve them

# Then push your changes
git push origin main
```

Why?

- Prevents push failures
- Gets collaborators' changes
- Reduces merge conflicts
- Keeps history clean

identitrain

Build Triggers
Automatic Builds

Builds run on a schedule (typically daily):

- Check catalog for course list
- Clone/update each course repository
- Build changed courses
- Publish outputs

Manual Builds

Trigger immediately via GKE console:

1. Navigate to Kubernetes Engine → Workloads
2. Find `courseops-builder` CronJob
3. Click "Run now"
4. Wait 5-15 minutes for completion

Monitoring Builds
Check Build Status

Via GKE Console:

- View CronJob history
- Check logs for errors
- See build duration
- Verify success/failure

Watch Logs

```
kubectl logs -n courseops <pod-name> -f
```

Look for:

- Course processing messages
- Slidev build output
- EPUB generation logs
- Error messages

Verifying Build Outputs

Once built, thoroughly check your course:

HTML Slides

- Navigate to `courses.identitrain.com/your-course-slug/`
- Click through all slides
- Verify images load
- Test Mermaid diagrams
- Check navigation works
- Verify layout and formatting

PDF Export

- Download PDF from course page
- Flip through all pages
- Verify images and diagrams
- Check page breaks and formatting

EPUB Verification

Check e-books in a reader:

Student Guide

- Open in Apple Books, Calibre, or similar
- Verify all chapters appear
- Check images display
- Read through notes sections
- Verify table of contents

Student Workbook

- Verify exercises are present
- Check formatting
- Test in multiple readers if possible

Common Issues:

- Missing images (check paths)
- Broken formatting (check Markdown syntax)
- Missing chapters (check slides.md references)

Common Issues and Fixes

Build Fails Completely

Symptoms: No course generated **Causes:**

- Syntax error in courses.yaml
- Invalid course code in catalog
- Repository access issues

Fix: Check build logs for error message

Images Don’t Display

Symptoms: Broken image icons **Causes:**

- Wrong file path
- Image not committed to Git
- Filename case mismatch

Fix: Verify path, commit image, check case

identitrain

More Common Issues

Mermaid Doesn’t Render

Symptoms: Shows as code block **Causes:**

- Syntax error in diagram
- Missing language identifier
- Indentation before code block

Fix: Test at mermaid.live, check backticks

Chapter Missing from Slides

Symptoms: Chapter doesn’t appear **Causes:**

- Not referenced in slides.md
- `hidden: true` in reference
- Syntax error in chapter reference

Fix: Check slides.md chapter references

Force Rebuild

Sometimes you need to rebuild everything:

Via Environment Variable

```
# In GKE CronJob configuration
FORCE_REBUILD=1
```

When to use:

- After theme updates
- When state file is corrupted
- To regenerate all outputs
- For troubleshooting

Best Practices Summary

For Commits

- □ Commit frequently (don’t batch days of work)
- □ Write descriptive messages
- □ Stage selectively for focused commits
- □ Review changes before committing (`git diff`)

For Pushing

- □ Pull before pushing
- □ Push after significant milestones
- □ Don’t push broken code
- □ Test locally when possible

For Builds

- □ Verify outputs after every build
- □ Check logs when builds fail
- □ Test changes locally first
- □ Use manual builds for rapid iteration

Publishing Complete!

What We've Covered

- ☐ Daily Git workflow (status, add, commit, push)
- ☐ Branching and merging
- ☐ Build triggers (automatic and manual)
- ☐ Monitoring builds and checking logs
- ☐ Verifying all output formats
- ☐ Troubleshooting common issues

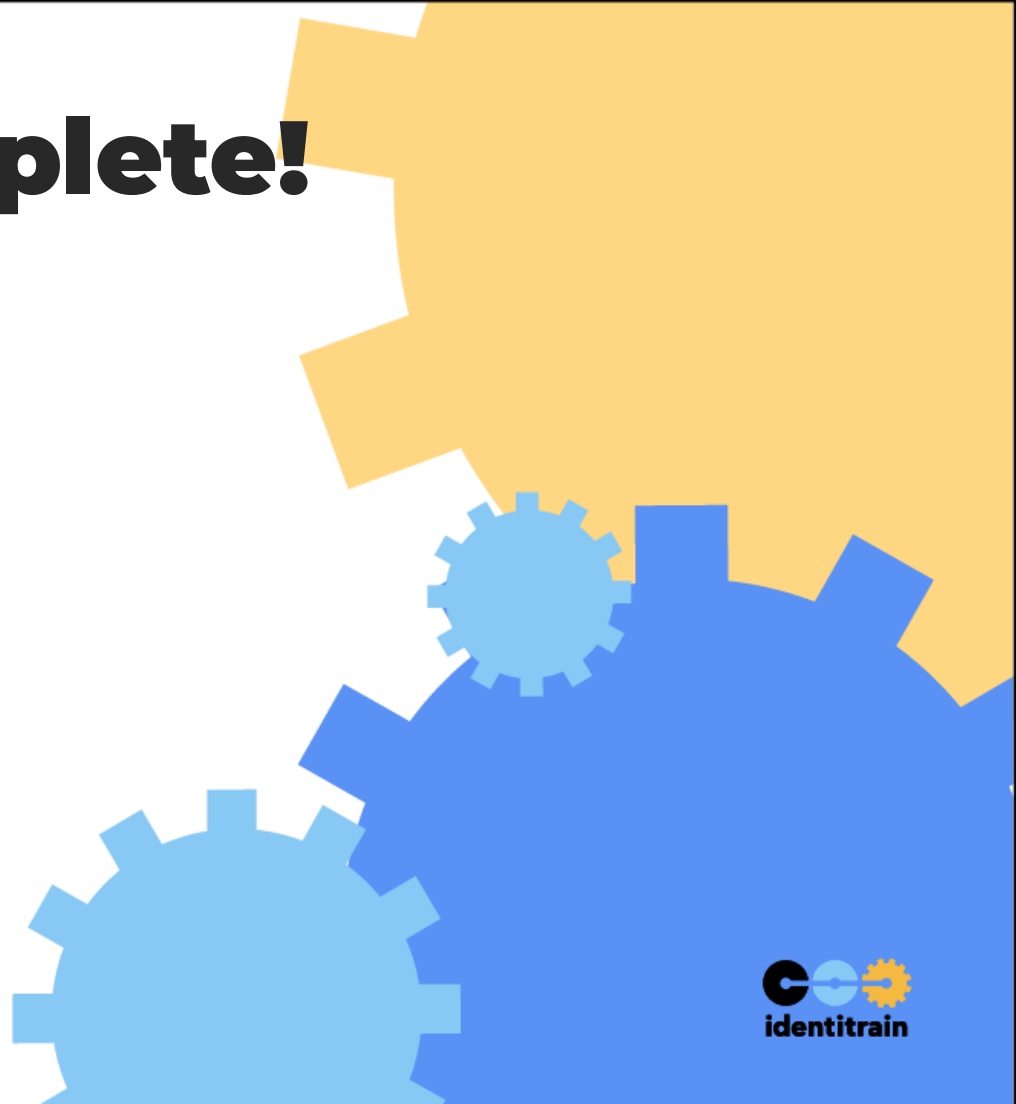
What's Next?

Course wrap-up and next steps for continued learning

identitrain

Course Complete!

You’re Ready to Develop Courses



identitrain

What You've Learned

Development Environment

- ☐ Installed and configured VSCode
- ☐ Set up essential extensions
- ☐ Learned keyboard shortcuts
- ☐ Customized workspace settings

Architecture Understanding

- ☐ Multi-repository structure
- ☐ Build pipeline stages
- ☐ Catalog configuration
- ☐ State management
- ☐ Kubernetes deployment

identitrain

What You’ve Learned (cont’d)

Course Creation

- ☐ Planned course structure
- ☐ Created directory layout
- ☐ Initialized Git repository
- ☐ Set up GitHub repository
- ☐ Created slides.md with frontmatter
- ☐ Wrote chapter files
- ☐ Added course to catalog

identitrain

What You’ve Learned (cont’d)

Content Creation

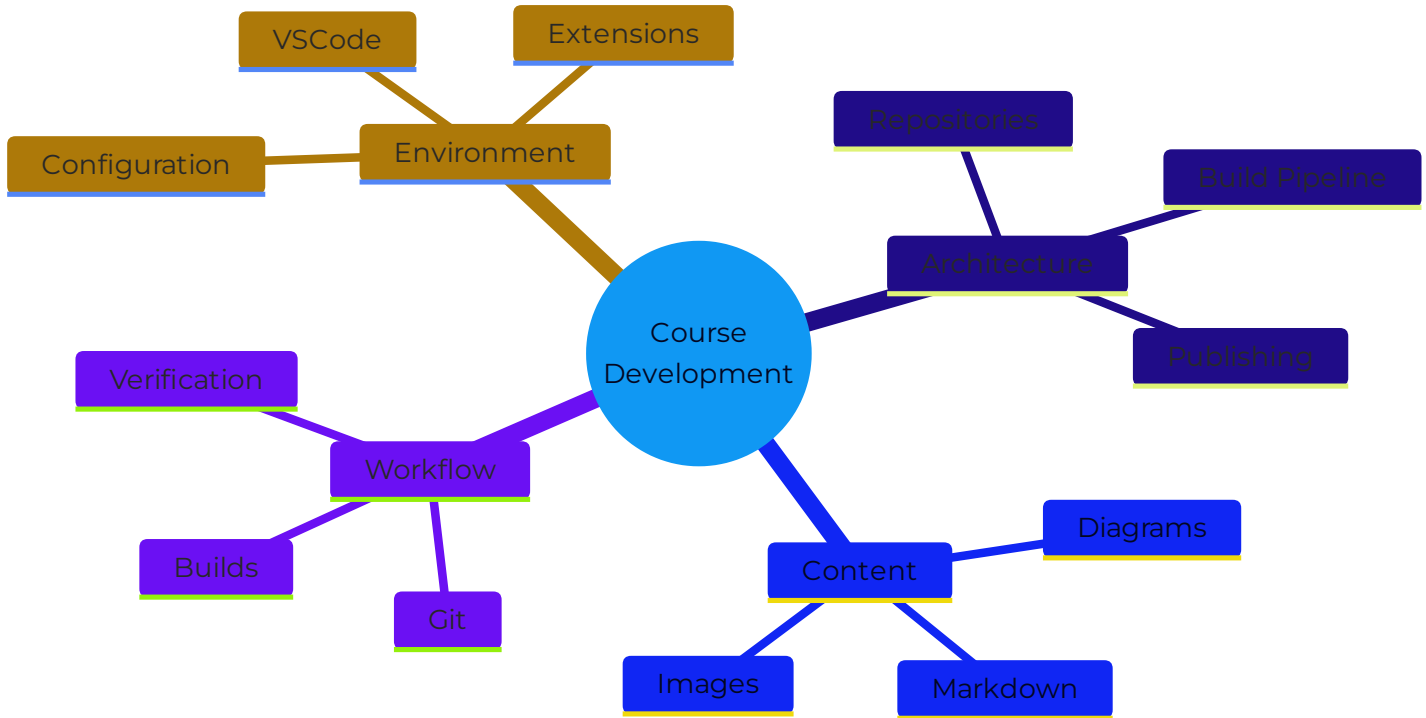
- ☐ Markdown syntax mastery
- ☐ Image integration
- ☐ Mermaid diagram creation
- ☐ Slide layouts and formatting
- ☐ Speaker notes and student guides
- ☐ Code blocks and syntax highlighting
- ☐ Tables and structure

What You’ve Learned (cont’d)

Publishing Workflow

- Daily Git operations
- Branching and merging
- Build triggering
- Output verification
- Troubleshooting
- Best practices

Your Complete Skill Set



From Here to Mastery
Next Steps

1. **Create Your First Real Course**

   - Pick a topic you know well
   - Apply everything you've learned
   - Start small (3-5 chapters)
   - Build and publish it

2. **Study Existing Courses**

   - Review IDT-AI-101 and IDT-GD-200
   - See how experts structure content
   - Learn from their techniques
   - Adapt patterns you like

identitrain

Resources for Continued Learning

Documentation

- **Course Developer Guide** - Complete reference in `docs/course-developer-guide/`
- **CLAUDE.md** - Technical details for the courseops system
- **README files** - Each repository's documentation

Example Courses

- **IDT-AI-101** - Comprehensive multi-day course
- **IDT-GD-200** - Technical deep-dive course
- **IDT-CD-001** - This course! (meta-reference)

External Resources

- Slidev Documentation
- Mermaid Documentation
- Markdown Guide

identitrain

Tips for Success

Content Creation

- **Write for your audience** - Match complexity to their level
- **Show, don't just tell** - Use examples and diagrams
- **Practice brevity** - Slides aren't essays
- **Test your content** - Have someone review it
- **Iterate constantly** - First draft is rarely final

Technical Skills

- **Commit early, commit often** - Fine-grained history
- **Test locally when possible** - Catch issues early
- **Learn Git deeply** - It's your safety net
- **Read error messages** - They usually tell you what's wrong
- **Ask for help** - Don't struggle alone

identitrain

Common Beginner Questions

Q: How long should a course be? A: 1 day = 40-80 slides typically. Quality over quantity.

Q: How many Mermaid diagrams should I include? A: Use diagrams where they clarify. Not every slide needs one.

Q: Should I create exercises? A: For hands-on courses, yes. For conceptual courses, optional.

Q: How often should I commit? A: After completing a logical unit of work (chapter, section, fix).

Q: What if I make a mistake? A: Git lets you revert. Don't be afraid to experiment.

identitrain

Giving and Receiving Feedback

When Reviewing Others' Courses

- **Be constructive** - Point out what works and what could improve
- **Be specific** - "This diagram is unclear" is better than "This is confusing"
- **Consider the audience** - Is it appropriate for the target students?
- **Suggest alternatives** - Don't just criticize, offer solutions

When Receiving Feedback

- **Don't take it personally** - Feedback improves the course
- **Ask clarifying questions** - Understand the concern
- **Consider the source** - Feedback from the target audience is most valuable
- **You decide** - Not all feedback must be implemented

identitrain

The Course Development Community

Collaboration Opportunities

- **Peer Review** - Exchange feedback with other developers
- **Shared Components** - Reuse diagrams, examples across courses
- **Best Practices** - Learn from each other's techniques
- **Tool Improvements** - Suggest CourseOps enhancements

Getting Help

- **Developer Guide** - First stop for questions
- **Team Members** - Experienced developers can help
- **GitHub Issues** - Report bugs or request features
- **Documentation PRs** - Improve docs for future developers

identitrain

Your First Course Challenge
Assignment

Create a 1-day introductory course on a topic you know well:

1. **Choose a topic** - Something you're expert in
2. **Plan structure** - 5-7 chapters
3. **Create repository** - Follow all setup steps
4. **Write content** - Include diagrams and examples
5. **Publish** - Add to catalog and build
6. **Share** - Get feedback from peers

Timeline: Complete within 2 weeks

Support: Available from course development team

identitrain

Continuous Improvement

Keep Learning

- **Stay updated** - CourseOps evolves, keep current
- **Experiment** - Try new layouts, diagram types
- **Analyze feedback** - What worked? What didn't?
- **Refine process** - Develop your personal workflow
- **Share knowledge** - Teach others what you learn

identitrain

Key Takeaways

Remember These Principles

- **Write once, output many** - Markdown generates all formats
- **Version control everything** - Git is your safety net
- **Build frequently** - Catch issues early
- **Verify thoroughly** - Check all outputs
- **Iterate constantly** - First draft is rarely perfect
- **Collaborate actively** - Better together than alone

identitrain

Thank You!
Congratulations on Completing This Course

You now have all the skills needed to:

- □ Set up a professional development environment
- □ Create courses from scratch
- □ Write rich, engaging content
- □ Publish and maintain courses
- □ Troubleshoot common issues
- □ Collaborate effectively

Now go create amazing training materials!

identitrain

Questions?

Feel free to reach out:

- **Course Development Team** - For technical support
- **GitHub Issues** - For bugs and feature requests
- **Developer Guide** - For reference and examples

Good luck with your course development journey!