Perfect! Now I'll create the complete OpenSCAD MCP server using the comprehensive documentation you provided. This will give the AI assistant deep OpenSCAD knowledge while maintaining the safe file operations.

SECTION 1: FILES TO CREATE

File 1: Dockerfile

```dockerfile
# Use Python slim image for a smaller footprint
FROM python:3.11-slim

# Set working directory
WORKDIR /app

# Set Python unbuffered mode
ENV PYTHONUNBUFFERED=1

# Copy requirements first for better caching
COPY requirements.txt .

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy the server code
COPY openscad_server.py .

# Create a non-root user and grant permissions to the app directory
RUN useradd -m -u 1000 mcpuser && \
    chown -R mcpuser:mcpuser /app

# Switch to the non-root user for security
USER mcpuser

# Run the server on container startup
CMD ["python", "openscad_server.py"]
```

File 2: requirements.txt

```

mcp[cli]>=1.2.0
```

File 3: openscad_server.py

```python
#!/usr/bin/env python3
```

```
"""Context-Aware OpenSCAD MCP Server - Provides comprehensive OpenSCAD knowledge and file management."""
import os
import sys
import logging
import glob
from mcp.server.fastmcp import FastMCP

# Configure logging to stderr for container logs
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    stream=sys.stderr
)
logger = logging.getLogger("openscad-server")

# Initialize MCP server
mcp = FastMCP("openscad")

# === COMPREHENSIVE OPENSCAD KNOWLEDGE BASE ===

SCAD_SYNTAX_RULES = """
📝 OpenSCAD Language: Foundational Syntax and Rules

I. GENERAL LANGUAGE CHARACTERISTICS
OpenSCAD is a text-based, programmatic solid 3D CAD modeler, often described as "The Programmers Solid 3D CAD Modeller". The language is primarily declarative and utilizes modules (procedures) and functions (mathematical calculations) to build complex models.

Core Syntax Elements:
• Variable Assignment: var = value; (valid in any scope since version 2015.03)
• Conditional Assignment: var = condition? value_if_true : value_if_false;
• Function Definition: function name(arg1, arg2) = expression;
• Module Definition: module name(arg1, arg2) { ... }
• Import: include <file.scad> (copies global variables)
• Import: use <file.scad> (modules/functions only)

Scope and Variable Rules:
• Immutability/Overriding: Variables act like override-able constants
• Scope Restriction: Assignments don't leak to outer scopes
• Last Assignment Rule: Last assignment applies everywhere in scope
• Case Sensitivity: Function/module names are case sensitive

II. FLOW CONTROL SYNTAX
• Conditional: if(condition1) { ... } else if(condition2) { ... } else { ... }
• For Loop (Range): for (i = [start : increment : end]) { ... }
• For Loop (List): for (i = [list_of_values]) { ... }
• Intersection Loop: intersection_for (i = [1:6]) { ... }
• List Comprehensions: list = [ for (i = range) if (condition) i ];
"""

SCAD_PRIMITIVES = """
🛠️ OpenSCAD Comprehensive Feature Reference - Primitives

3D PRIMITIVES:
• cube(size, center) or cube([w,d,h], center) - Rectangular prism
• sphere(r=radius) or sphere(d=diameter) - Spherical object
• cylinder(h, r|d, center) or cylinder(h, r1|d1, r2|d2, center) - Cylinder/frustum
• polyhedron(points, faces, convexity) - Complex 3D shape from points/faces

2D PRIMITIVES (lie in XY plane, require extrusion):
• circle(r=radius) or circle(d=diameter) - Planar circle
• square(size, center) or square([w,h], center) - Square/rectangle
• polygon([points], [paths]) - Planar shape from points
• text(t, size, font,...) - 2D text geometry (requires extrusion)
• projection(cut=true) - Projects 3D object to XY plane
"""

SCAD_OPERATIONS = """
⚙️ OpenSCAD CSG Operations and Transformations

CONSTRUCTIVE SOLID GEOMETRY (CSG):
• union() { obj1; obj2; } - Combines objects into single unified object
• difference() { base_obj; subtract_obj1; subtract_obj2; } - Removes subsequent objects from
first
• intersection() { obj1; obj2; } - Creates object from shared volume only

TRANSFORMATIONS:
• translate([x,y,z]) { ... } - Moves child object by vector
• rotate([x,y,z]) { ... } or rotate(angle, [x,y,z]) { ... } - Rotates child object
• scale([x,y,z]) { ... } - Resizes along X, Y, and Z axes
• resize([x,y,z], auto=false, convexity) - Non-uniform scaling to fit dimensions
• mirror([x,y,z]) { ... } - Mirrors across plane defined by normal vector
• multmatrix(m) { ... } - Applies custom 4x4 transformation matrix

GEOMETRY OPERATIONS:
• hull() { obj1; obj2; } - Creates convex hull of all child objects
• minkowski(convexity) { obj1; obj2; } - Creates Minkowski sum
• offset(r|delta, chamfer) - Offsets edges of 2D shape or 3D surface
• linear_extrude(height, twist,...) - Extrudes 2D shape along straight path
• rotate_extrude(angle,...) - Rotates 2D shape around Z-axis
• surface(file="...", center, convexity) - Creates 3D surface from height-map
"""

SCAD_SPECIAL_VARS = """
🎛️ OpenSCAD Special Variables and Modifiers
```

CIRCLE RESOLUTION VARIABLES:
- $fn = 0 - Fragments Number (sets segment count, overrides $fa/$fs if >0)
- $fa = 12 - Fragment Angle (minimum angle in degrees for segments)
- $fs = 2 - Fragment Size (minimum size for line segments)

DEBUGGING AND RENDERING MODIFIERS:
- # - Debug/Highlight: Shows object in transparent pink for visualization
- % - Background/Transparent: Shows in gray but ignores for CSG operations
- ! - Root/Show Only: Uses marked subtree as temporary design root
- * - Disable: Completely ignores marked subtree

OTHER SPECIAL VARIABLES:
- $children - Number of child nodes passed to current module
- $preview - Boolean: true in F5 preview, false in F6 render
- $t - Current animation step value for animations

UTILITY FUNCTIONS:
- echo("Variable Value:", my_var) - Diagnostic output to console
- assert(value > 0, "Value must be positive") - Condition checking
- children(0) - Returns first child object passed to module
- render() { complicated_object; } - Forces rendering operation

"""

SCAD_BEST_PRACTICES = """

🏆 OpenSCAD Best Practices and Common Patterns

PARAMETERIZED DESIGN:
- Use variables for all dimensions to enable easy modifications
- Group related parameters at the top of files
- Use meaningful variable names (wall_thickness vs wt)

MODULE ORGANIZATION:
- Break complex designs into logical modules
- Use descriptive module names that indicate purpose
- Document module parameters and expected behavior

PERFORMANCE OPTIMIZATION:
- Use $fn sparingly - high values dramatically increase render time
- Prefer $fa and $fs for adaptive resolution
- Use render() for complex recursive operations
- Avoid excessive difference() operations with many children

DEBUGGING TECHNIQUES:
- Use # modifier to visualize intermediate steps
- Employ % to see reference geometry without affecting CSG
- Use echo() to output variable values during rendering
- Test modules in isolation before integration

```
STL EXPORT CONSIDERATIONS:
• Ensure manifold geometry (no holes or non-solid objects)
• Check normals are consistent for 3D printing
• Use sufficient resolution for intended print size
• Verify dimensions match expected real-world units
"""

# Category mapping for easy reference
SCAD_CATEGORIES = {
    "syntax": SCAD_SYNTAX_RULES,
    "primitives": SCAD_PRIMITIVES,
    "operations": SCAD_OPERATIONS,
    "variables": SCAD_SPECIAL_VARS,
    "bestpractices": SCAD_BEST_PRACTICES,
    "3d": "3D Primitives: cube(), sphere(), cylinder(), polyhedron()",
    "2d": "2D Primitives: circle(), square(), polygon(), text()",
    "transformations": "Transformations: translate(), rotate(), scale(), mirror(), resize()",
    "boolean": "Boolean Operations: union(), difference(), intersection()",
    "extrusions": "Extrusions: linear_extrude(), rotate_extrude()"
}

# === MCP TOOLS ===

@mcp.tool()
async def list_files(file_extension: str = "") -> str:
    """Lists files in the current directory, optionally filtering by extension."""
    logger.info(f"Executing list_files with extension '{file_extension}'")
    try:
        pattern = f"*.{file_extension}" if file_extension.strip() else "*"
        files = glob.glob(pattern)
        if not files:
            return f"📂 No files found with pattern '{pattern}'."
        scad_files = [f for f in files if f.endswith('.scad')]
        other_files = [f for f in files if not f.endswith('.scad')]

        output = "📂 Files in directory:"
        if scad_files:
            output += f"\n\n📐 OpenSCAD Files:\n- " + "\n- ".join(scad_files)
        if other_files:
            output += f"\n\n📄 Other Files:\n- " + "\n- ".join(other_files)
        return output
    except Exception as e:
        logger.error(f"Error listing files: {e}")
        return f"❌ Error: Could not list files. {str(e)}"

@mcp.tool()
async def read_file(filename: str = "") -> str:
    """Reads and returns the content of a specified file."""
    logger.info(f"Executing read_file for '{filename}'")
    if not filename.strip():
        return "❌ Error: Filename is required."
    try:
        with open(filename, 'r') as f:
            content = f.read()

        file_info = f"📄 File: {filename} ({len(content)} characters)"
        if filename.endswith('.scad'):
            file_info += " 📐 OpenSCAD"

        return f"{file_info}\n---\n{content}\n---"
    except FileNotFoundError:
        return f"❌ Error: File '{filename}' not found."
    except Exception as e:
        logger.error(f"Error reading file '{filename}': {e}")
        return f"❌ Error: Could not read file. {str(e)}"

@mcp.tool()
async def write_file(filename: str = "", content: str = "", overwrite: str = "false") -> str:
    """Creates a new file with content, with an option to overwrite if it exists."""
    logger.info(f"Executing write_file for '{filename}'")
    if not filename.strip():
        return "❌ Error: Filename is required."

    should_overwrite = overwrite.strip().lower() == 'true'

    if os.path.exists(filename) and not should_overwrite:
        return f"⚠️ Error: File '{filename}' already exists. To overwrite, set overwrite to 'true'."

    try:
        with open(filename, 'w') as f:
            f.write(content)
        action = "overwritten" if should_overwrite else "created"
        file_type = " 📐 OpenSCAD" if filename.endswith('.scad') else ""
        return f"✅ Success: File '{filename}'{file_type} was {action}."
    except Exception as e:
        logger.error(f"Error writing file '{filename}': {e}")
        return f"❌ Error: Could not write to file. {str(e)}"

@mcp.tool()
async def append_to_file(filename: str = "", content: str = "") -> str:
    """Appends content to the end of an existing file."""
    logger.info(f"Executing append_to_file for '{filename}'")
    if not filename.strip() or not content.strip():
        return "❌ Error: Both filename and content are required."
    try:
        with open(filename, 'a') as f:
            f.write("\n" + content)
        file_type = " 📐 OpenSCAD" if filename.endswith('.scad') else ""
        return f"✅ Success: Content appended to '{filename}'{file_type}."
    except FileNotFoundError:
        return f"❌ Error: File '{filename}' not found."
    except Exception as e:
        logger.error(f"Error appending to file '{filename}': {e}")
        return f"❌ Error: Could not append to file. {str(e)}"

@mcp.tool()
async def get_scad_syntax() -> str:
    """Returns comprehensive OpenSCAD syntax rules and language characteristics."""
    logger.info("Executing get_scad_syntax")
    return SCAD_SYNTAX_RULES

@mcp.tool()
async def get_scad_reference(category: str = "") -> str:
    """Returns detailed OpenSCAD reference for specific categories."""
    logger.info(f"Executing get_scad_reference with category '{category}'")
    cat = category.strip().lower()

    if cat in SCAD_CATEGORIES:
        category_names = {
            "syntax": "📝 Syntax and Rules",
            "primitives": "🛠️ Primitives",
            "operations": "⚙️ Operations and Transformations",
            "variables": "🎛️ Special Variables and Modifiers",
            "bestpractices": "🏆 Best Practices"
        }
        title = category_names.get(cat, cat.upper())
        return f"{title}\n{SCAD_CATEGORIES[cat]}"
    elif cat in ["3d", "2d", "transformations", "boolean", "extrusions"]:
        return f"🔧 {cat.upper()} Functions:\n{SCAD_CATEGORIES[cat]}"
    elif not cat:
        output = "📚 Available OpenSCAD Reference Categories:\n\n"
        output += "• syntax - Language syntax and rules\n"
        output += "• primitives - 2D and 3D primitive shapes\n"
        output += "• operations - CSG operations and transformations\n"
        output += "• variables - Special variables and modifiers\n"
        output += "• bestpractices - Design patterns and optimization\n"
        output += "• 3d - Quick 3D primitive reference\n"
        output += "• 2d - Quick 2D primitive reference\n"
        output += "• transformations - Quick transformation reference\n"
        output += "• boolean - Quick boolean operations reference\n"
        output += "• extrusions - Quick extrusion operations reference\n"
        return output
    else:
        available = ", ".join(SCAD_CATEGORIES.keys())
        return f"❌ Category '{cat}' not found. Available: {available}"

@mcp.tool()
async def scad_quick_help(topic: str = "") -> str:
    """Provides quick help for common OpenSCAD topics and functions."""
    logger.info(f"Executing scad_quick_help for '{topic}'")

    quick_reference = {
        "cube": "cube(size, center) - Creates cube/rectangular prism\nExample: cube([10,20,5],
center=true);",
        "sphere": "sphere(r=radius) or sphere(d=diameter) - Creates sphere\nExample:
sphere(r=10, $fn=50);",
        "cylinder": "cylinder(h, r|d, center) - Creates cylinder/frustum\nExample: cylinder(h=20,
r1=10, r2=5, center=true);",
        "translate": "translate([x,y,z]) { ... } - Moves child object\nExample: translate([5,0,0])
cube(10);",
        "rotate": "rotate([x,y,z]) { ... } - Rotates child object\nExample: rotate([0,0,45])
cube(10);",
        "difference": "difference() { base; subtract1; subtract2; } - Boolean
subtraction\nExample: difference() { cube(10); cylinder(h=15, r=3); }",
        "module": "module name(params) { ... } - Defines reusable component\nExample:
module box(size) { cube(size); }",
        "extrude": "linear_extrude(height, twist, ...) { 2d_shape; } - Extrudes 2D to
3D\nExample: linear_extrude(10) circle(5);",
        "variables": "Special variables: $fn, $fa, $fs for resolution\nModifiers: # (debug), %
(background), ! (root), * (disable)"
    }

    if not topic.strip():
        return "🔧 Quick OpenSCAD Reference - Available topics:\n- " + "\n-
".join(quick_reference.keys())

    topic_lower = topic.strip().lower()
    if topic_lower in quick_reference:
        return f"🔧 {topic.capitalize()}:\n{quick_reference[topic_lower]}"
    else:
        available = ", ".join(quick_reference.keys())
        return f"❌ Topic '{topic}' not found. Available: {available}"

# === SERVER STARTUP ===
if __name__ == "__main__":
    logger.info("Starting Context-Aware OpenSCAD MCP server...")
    logger.info("OpenSCAD knowledge base loaded with comprehensive documentation")
    try:
        mcp.run(transport='stdio')
    except Exception as e:
        logger.error(f"Server error: {e}", exc_info=True)
```

~~~
        sys.exit(1)
```
~~~

File 4: readme.md

~~~
```markdown
# OpenSCAD MCP Server

A comprehensive Model Context Protocol (MCP) server that provides deep OpenSCAD knowledge and safe file management for 3D modeling workflows.

## Purpose

This server acts as an expert OpenSCAD reference and project assistant, providing AI assistants with authoritative OpenSCAD documentation and safe file operations to accelerate 3D design workflows.

## Features

### File System Tools
- **`list_files(file_extension)`** - View directory contents with OpenSCAD file highlighting
- **`read_file(filename)`** - Read files to understand current project state
- **`write_file(filename, content, overwrite)`** - Create or update files with safety checks
- **`append_to_file(filename, content)`** - Add content to existing files

### Comprehensive OpenSCAD Knowledge
- **`get_scad_syntax()`** - Complete language syntax, rules, and flow control
- **`get_scad_reference(category)`** - Detailed reference by category:
  - `syntax` - Language characteristics and structure
  - `primitives` - 2D and 3D primitive shapes
  - `operations` - CSG operations and transformations
  - `variables` - Special variables and debugging modifiers
  - `bestpractices` - Design patterns and optimization
- **`scad_quick_help(topic)`** - Quick function reference for common operations

## Knowledge Coverage

Based on comprehensive OpenSCAD documentation, this server provides:

### Language Fundamentals
- Variable assignment and scoping rules
- Module and function definitions
- Flow control (if/else, for loops, list comprehensions)
- Import directives (include vs use)

### Geometry Primitives
- 3D: cube, sphere, cylinder, polyhedron
- 2D: circle, square, polygon, text
~~~

- Extrusions: linear_extrude, rotate_extrude

### Operations & Transformations
- CSG: union, difference, intersection
- Geometric: translate, rotate, scale, mirror, resize
- Advanced: hull, minkowski, offset

### Special Features
- Resolution control: $fn, $fa, $fs
- Debugging modifiers: #, %, !, *
- Animation variables: $t
- Utility functions: echo, assert, children

## Strategic Usage

The server enables AI assistants to follow this optimal workflow:

1. **Orient** - Use `list_files()` to understand project structure
2. **Analyze** - Use `read_file()` to examine existing SCAD files
3. **Reference** - Access comprehensive documentation via `get_scad_reference()`
4. **Implement** - Generate OpenSCAD code using authoritative knowledge
5. **Modify** - Safely update files with proper validation

## Security

- Runs as non-root user in Docker container
- File operations include overwrite protection
- Input validation on all parameters
- No external network dependencies
- Comprehensive error handling

## Quick Start

```bash
# Build the image
docker build -t openscad-mcp-server .

# The server will be available through Claude Desktop MCP integration
```

Example Queries

· "Show me OpenSCAD syntax rules"
· "What transformation functions are available?"
· "How do I create a parameterized module?"
· "Explain the difference between union and difference"
· "What are the special variables for resolution control?"

This server transforms AI assistants into expert OpenSCAD consultants with instant access to comprehensive 3D modeling knowledge.

~~~

File 5: LLM_GUIDE.md

```markdown
# LLM Guide: Using the OpenSCAD MCP Server

This guide outlines the optimal strategy for leveraging the comprehensive OpenSCAD knowledge base and file operations to provide expert 3D modeling assistance.

## Core Philosophy: Context-First Design

You are an OpenSCAD expert consultant. Use the server's knowledge base as your primary reference to ensure accuracy and prevent hallucination. The server provides authoritative OpenSCAD documentation - leverage it extensively.

## Optimal Workflow Pattern

### 1. **Context Gathering Phase**
Always start by understanding the user's project context:

```python
# Check project structure
files = list_files("scad")
# Read existing files for context
current_design = read_file("project.scad")
# Reference documentation as needed
syntax = get_scad_reference("syntax")
~~~

2. Knowledge Consultation Phase

Before generating code, consult the appropriate reference sections:

· For syntax questions: get_scad_syntax()
· For shape references: get_scad_reference("primitives")
· For operations: get_scad_reference("operations")
· For best practices: get_scad_reference("bestpractices")
· Quick lookups: scad_quick_help("topic")

3. Implementation Phase

Generate OpenSCAD code using the authoritative knowledge, then safely apply changes:

```python
```

```
# For new files
write_file("new_design.scad", generated_code)
# For modifications (after reading current state)
write_file("existing.scad", updated_code, overwrite="true")
# For additions
append_to_file("library.scad", new_module)
```

Scenario-Based Strategies

Scenario 1: New Project Creation

User: "Create a parameterized box with screw holes"

Your Approach:

1. Use get_scad_reference("bestpractices") for parameterized design patterns
2. Use get_scad_reference("primitives") for cube/cylinder references
3. Use get_scad_reference("operations") for difference() operation
4. Generate well-structured, parameterized code
5. Use write_file() to create the new SCAD file

Scenario 2: Modifying Existing Design

User: "Add fillets to the edges of my box in box.scad"

Your Approach:

1. ALWAYS use read_file("box.scad") first to understand current implementation
2. Use get_scad_reference("operations") to review offset() and minkowski()
3. Use scad_quick_help("difference") for boolean operation syntax
4. Generate modifications that preserve existing structure
5. Use write_file() with overwrite="true" to apply changes

Scenario 3: Debugging Assistance

User: "My model has holes and isn't manifold"

Your Approach:

1. Read the problematic file: read_file("problem.scad")
2. Consult get_scad_reference("bestpractices") for manifold geometry guidelines
3. Use get_scad_reference("variables") to explain debugging modifiers (#, %)
4. Suggest specific fixes based on OpenSCAD best practices

Knowledge Base Utilization Patterns

When to Use Which Reference

Use Case Recommended Tools
Syntax questions get_scad_syntax(), get_scad_reference("syntax")
Shape reference get_scad_reference("primitives"), scad_quick_help()
Transformation help get_scad_reference("operations")
Performance issues get_scad_reference("bestpractices")
Debugging get_scad_reference("variables") for modifiers
Quick function lookup scad_quick_help("function_name")

Common Reference Patterns

For complex designs:

```python
# Get comprehensive guidance
best_practices = get_scad_reference("bestpractices")
operations = get_scad_reference("operations")
```

For specific operations:

```python
# Quick function reference
extrude_help = scad_quick_help("extrude")
boolean_help = scad_quick_help("difference")
```

Safety-First File Operations

File Modification Protocol

1. Always read first: Never modify without understanding current state
2. Use overwrite protection: Require explicit overwrite="true" for existing files
3. Preserve structure: Maintain existing comments and organization
4. Test incrementally: Suggest small, testable changes

Example Safe Modification

```python
# UNSAFE: Modifying without context
write_file("design.scad", new_code, overwrite="true")

# SAFE: Context-aware modification
current = read_file("design.scad")
# Analyze current structure, then...
write_file("design.scad", improved_code, overwrite="true")
```

Advanced Assistance Patterns

Educational Approach

When users are learning OpenSCAD:

- Use get_scad_reference() to provide structured learning
- Explain concepts using authoritative documentation
- Suggest scad_quick_help() for immediate syntax questions

Optimization Guidance

For performance issues:

- Reference resolution variables ($fn, $fa, $fs) from get_scad_reference("variables")
- Apply best practices from get_scad_reference("bestpractices")
- Suggest specific optimizations based on documented patterns

Debugging Strategy

1. Identify the issue type (manifold, performance, syntax)
2. Consult the relevant knowledge base section
3. Use debugging modifiers (#, %) for visualization
4. Apply documented solutions

Response Formatting Guidelines

When Providing Code Examples

~~~
I'll help you with that. Let me check the OpenSCAD documentation first:

[Consult get_scad_reference() or scad_quick_help()]

Based on the OpenSCAD documentation, here's how you can implement this:

```openscad
// Well-commented code example
module your_design() {
  // Implementation using best practices
}
```

Key points from the documentation:

· Point 1 from reference
· Point 2 from reference
~~~

```

### When Answering Questions
```

According to the OpenSCAD documentation:

[Reference from knowledge base]

In practice, this means:

· Practical implication 1
· Practical implication 2

```

## Continuous Learning

The server's knowledge base is comprehensive and authoritative. Always prefer referencing it over relying solely on your training data for OpenSCAD-specific questions. This ensures users receive accurate, up-to-date information based on the actual OpenSCAD language specification.

By following these guidelines, you'll provide expert-level OpenSCAD assistance that's both accurate and practical, leveraging the full power of the comprehensive knowledge base while maintaining safe file operations.
```

SECTION 2: INSTALLATION INSTRUCTIONS FOR THE USER

Step 1: Save the Files

```bash
# Create project directory
mkdir openscad-mcp-server
cd openscad-mcp-server

# Save all 5 files in this directory
```

Step 2: Build Docker Image

```bash
docker build -t openscad-mcp-server .
```

Step 3: Create Custom Catalog

```bash
# Create catalogs directory if it doesn't exist
mkdir -p ~/.docker/mcp/catalogs

# Create or edit custom.yaml
nano ~/.docker/mcp/catalogs/custom.yaml
```

Add this entry to custom.yaml:

```yaml
version: 2
name: custom
displayName: Custom MCP Servers
registry:
  openscad:
    description: "Comprehensive OpenSCAD knowledge base and file management for 3D modeling"
    title: "OpenSCAD Assistant"
    type: server
    dateAdded: "2025-01-15T00:00:00Z"
    image: openscad-mcp-server:latest
    ref: ""
    readme: ""
    toolsUrl: ""
    source: ""
    upstream: ""
    icon: ""
    tools:
      - name: list_files
      - name: read_file
      - name: write_file
      - name: append_to_file
      - name: get_scad_syntax
      - name: get_scad_reference
      - name: scad_quick_help
    secrets: []
    metadata:
      category: productivity
      tags:
        - openscad
        - 3d-modeling
        - cad
        - 3d-printing
      license: MIT
      owner: local
```

Step 4: Update Registry

```bash
# Edit registry file
nano ~/.docker/mcp/registry.yaml
```

Add this entry under the existing registry: key:

```yaml
registry:
  # ... existing servers ...
  openscad:
    ref: ""
```

Step 5: Configure Claude Desktop

Find your Claude Desktop config file:

- macOS: ~/Library/Application Support/Claude/claude_desktop_config.json
- Windows: %APPDATA%\Claude\claude_desktop_config.json
- Linux: ~/.config/Claude/claude_desktop_config.json

Edit the file and add your custom catalog to the args array:

```json
{
  "mcpServers": {
    "mcp-toolkit-gateway": {
      "command": "docker",
      "args": [
        "run",
        "-i",
        "--rm",
        "-v", "/var/run/docker.sock:/var/run/docker.sock",
        "-v", "/Users/your_username/.docker/mcp:/mcp",
        "docker/mcp-gateway",
        "--catalog=/mcp/catalogs/docker-mcp.yaml",
        "--catalog=/mcp/catalogs/custom.yaml",
        "--config=/mcp/config.yaml",
        "--registry=/mcp/registry.yaml",
        "--tools-config=/mcp/tools.yaml",
        "--transport=stdio"
      ]
    }
  }
}
```

~~~
```
~~~

Replace /Users/your_username with your actual home directory path.

Step 6: Restart Claude Desktop

1. Quit Claude Desktop completely
2. Start Claude Desktop again
3. Your OpenSCAD tools should appear!

Step 7: Test Your Server

```bash
# Verify it appears in the list
docker mcp server list
```

Usage Examples

Once installed, you can ask Claude:

- "Show me the OpenSCAD syntax rules"
- "What functions are available for transformations?"
- "Create a new SCAD file for a parametric box"
- "Read my existing project.scad and suggest improvements"
- "How do I use the hull() operation?"

Your comprehensive OpenSCAD MCP server is now ready! It provides deep OpenSCAD knowledge combined with safe file operations to accelerate your 3D modeling workflow.