TOSCA Architecture Documentation - PDF Generation Guide

Architecture Diagrams

Last Updated: 2025-11-05 **Purpose:** Best practices for converting architecture documentation to high-quality PDFs

---

Problem Statement

The architecture documentation markdown files contain ASCII box drawing characters (┌─┐│└┘├┤┬┴┼) for architecture diagrams. When converting to PDF using `pandoc` with `wkhtmltopdf`, these characters render poorly due to:

1. **Font issues:** Box drawing characters require specific Unicode fonts (DejaVu Sans Mono, Courier New, etc.)
2. **Character encoding:** wkhtmltopdf may not preserve UTF-8 encoding correctly
3. **Monospace alignment:** Box drawings require precise monospace alignment
4. **Print scaling:** PDF rendering can distort character spacing

---

Recommended Solution: Replace ASCII Diagrams with Images

Strategy

Replace ASCII diagrams with PlantUML-generated images for PDF output while preserving ASCII diagrams for quick markdown readability.

Why This Approach?

1. **Professional appearance:** PNG/SVG images render perfectly in PDFs
2. **Scalability:** Vector graphics (SVG) scale without quality loss
3. **Maintainability:** PlantUML diagrams are version-controlled as code
4. **Dual-format support:** Keep ASCII for quick markdown viewing, use images for PDF generation
5. **Medical device compliance:** Professional documentation for regulatory submissions (FDA, ISO 13485)

Implementation Guide

Step 1: Verify PlantUML Diagram Availability

You already have PlantUML diagrams in `/docs/architecture/diagrams/`:

Available Diagrams: - `TOSCA System Context.png` (C4 context diagram) - `TOSCA Container Diagram.png` (C4 container diagram) - `TOSCA Component Diagram - Application Core.png` (C4 component diagram - core) - `TOSCA Component Diagram - Hardware Abstraction Layer.png` (C4 component diagram - HAL) - `TOSCA Data Architecture.png` (Two-tier logging strategy) - `TOSCA Data Flow Diagram.png` (System data flow) - `TOSCA Database Schema ERD.png` (Database entity-relationship diagram) - `TOSCA Treatment Workflow Sequence.png` (Treatment sequence diagram)

Format Options: - PNG (for PDF embedding): `/docs/architecture/diagrams/output/png/` - SVG (for web/scaling): `/docs/architecture/diagrams/output/svg/`

Step 2: Map ASCII Diagrams to PlantUML Images

Markdown File	Section	ASCII Diagram	Replacement Image
`01_system_overview.md`	High-Level Architecture	System components box diagram	`TOSCA Container Diagram.png`
`02_database_schema.md`	Database Schema	ERD ASCII diagram	`TOSCA Database Schema ERD.png`
`03_safety_system.md`	Interlock Architecture	Safety interlock hierarchy	`TOSCA Component Diagram - Application Core.png` (Safety Manager section)
`04_treatment_protocols.md`	Protocol Execution Flow	Protocol engine state diagram	`TOSCA Treatment Workflow Sequence.png`

`06_protocol_builder.md`	UI Workflow	Protocol builder
		
		component diagram
			`TOSCA Component Diagram -`
			
			`Application Core.png`

`07_safety_watchdog.md`	Watchdog Architecture	Watchdog communication diagram	Create new PlantUML diagram (if needed)
`08_security_architecture.md`	Security Layers	Security architecture diagram	`TOSCA Data Architecture.png`
`09_test_architecture.md`	Test Structure	Test layer hierarchy	Create new PlantUML diagram (if needed)
`10_concurrency_model.md`	Thread Model	Thread communication diagram	Create new PlantUML diagram (if needed)

Step 3: Update Markdown Files with Conditional Diagram Rendering

Approach: Use HTML comments to conditionally include ASCII (for markdown) or images (for PDF).

Pattern:

~~~
## Section Title

<!-- BEGIN ASCII (Markdown viewing) -->
```text
┌──────────────────────────┐
│   Safety Manager         │
│   (Master Controller)    │
└──────────────────────────┘
             │
             │
     ┌───────┼────────┐
     │       │        │
~~~

Explanation: The safety manager coordinates...

~~~
**Alternative: Pandoc Filter Approach (Recommended)**

Use pandoc filters to automatically replace code blocks with images during PDF generation:

```bash
# Convert with filter
pandoc 03_safety_system.md -o 03_safety_system.pdf \
  --pdf-engine=wkhtmltopdf \
  --lua-filter=replace-ascii-diagrams.lua
~~~

Lua Filter (`replace-ascii-diagrams.lua`):

```lua
-- Replace code blocks containing box drawing chars with images
function CodeBlock(block)
  local text = block.text

  -- Check if code block contains box drawing characters
  if text:match('[┌┐└┘│─├┤┬┴┼]') then
    -- Replace with image based on parent document
    local image_path = "diagrams/output/png/diagram.png"
    return pandoc.Para({pandoc.Image({}, image_path, "Architecture Diagram")})
  end

  return block
end
```

Step 4: Create Missing PlantUML Diagrams

For diagrams not yet available as PlantUML sources:

Example: Safety Watchdog Architecture

Create `/docs/architecture/diagrams/watchdog-architecture.puml`:

```
@startuml TOSCA Watchdog Architecture
!include https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4_Component.puml

title TOSCA Safety Watchdog Architecture

Component(safety_manager, "Safety Manager", "Python", "Master safety controller")
Component(watchdog, "Safety Watchdog", "Python Thread", "Heartbeat sender")
Component(arduino, "Arduino Firmware", "C++", "Watchdog timer")

Component(laser, "Laser Controller", "Python HAL", "Power control")
Component(tec, "TEC Controller", "Python HAL", "Temperature control")

Rel(safety_manager, watchdog, "Monitors", "")
Rel(watchdog, arduino, "Sends heartbeat (500ms)", "Serial")
Rel(arduino, laser, "Hardware interlock", "GPIO")
Rel(arduino, tec, "Hardware interlock", "GPIO")

note right of watchdog
  Watchdog sends heartbeat every 500ms
  Timeout: 1000ms
  Failure triggers hardware E-stop
end note

@enduml
```

Generate diagram:

```
cd /docs/architecture/diagrams
java -jar plantuml.jar -tpng watchdog-architecture.puml -o output/png
java -jar plantuml.jar -tsvg watchdog-architecture.puml -o output/svg
```

Step 5: Improved Pandoc PDF Generation

Option 1: Use CSS for Better Font Rendering (wkhtmltopdf)

Create `/docs/architecture/pdf-styles.css`:

```
/* Force monospace font for code blocks */
pre, code {
  font-family: "DejaVu Sans Mono", "Courier New", "Consolas", monospace !important;
  font-size: 10pt;
  line-height: 1.2;
}

/* Preserve whitespace and prevent wrapping */
pre {
  white-space: pre;
  overflow-x: auto;
  page-break-inside: avoid;
}

/* Image sizing for diagrams */
img {
  max-width: 100%;
  height: auto;
  display: block;
  margin: 20px auto;
}

/* Page margins */
@page {
  margin: 2cm;
}

/* Headers and footers */
h1, h2, h3 {
  page-break-after: avoid;
}
```

Convert with CSS:

```
pandoc 03_safety_system.md -o 03_safety_system.pdf \
  --pdf-engine=wkhtmltopdf \
  --css=pdf-styles.css \
  --metadata title="TOSCA Safety System Architecture"
```

Option 2: Use LaTeX Engine (Better Quality)

```
# Install LaTeX engine (if not already installed)
# Ubuntu: sudo apt install texlive-xetex
# macOS: brew install --cask mactex

# Convert with XeLaTeX (supports Unicode box drawing)
pandoc 03_safety_system.md -o 03_safety_system.pdf \
  --pdf-engine=xelatex \
  --variable mainfont="DejaVu Sans Mono" \
  --variable geometry:margin=1in
```

Option 3: Two-Pass Conversion (HTML → PDF)

```
# Pass 1: Markdown to HTML with embedded images
pandoc 03_safety_system.md -o 03_safety_system.html \
  --standalone \
  --self-contained \
  --css=pdf-styles.css

# Pass 2: HTML to PDF with wkhtmltopdf
wkhtmltopdf \
  --enable-local-file-access \
  --encoding utf-8 \
  --dpi 300 \
  03_safety_system.html 03_safety_system.pdf
```

Recommended Workflow

For Single File Conversion

```
#!/bin/bash
# convert-to-pdf.sh

MARKDOWN_FILE="$1"
OUTPUT_PDF="${MARKDOWN_FILE%.md}.pdf"

# Check if file exists
if [ ! -f "$MARKDOWN_FILE" ]; then
  echo "Error: File not found: $MARKDOWN_FILE"
  exit 1
fi

# Convert using XeLaTeX (best quality)
pandoc "$MARKDOWN_FILE" -o "$OUTPUT_PDF" \
  --pdf-engine=xelatex \
  --variable mainfont="DejaVu Sans Mono" \
  --variable geometry:margin=1in \
  --toc \
  --toc-depth=3 \
  --number-sections \
  --metadata title="TOSCA Architecture Documentation" \
  --metadata date="$(date +%Y-%m-%d)"

echo "Generated: $OUTPUT_PDF"
```

Usage:

```
./convert-to-pdf.sh 03_safety_system.md
```

For Batch Conversion (All Architecture Docs)

```bash
#!/bin/bash
# convert-all-to-pdf.sh

OUTPUT_DIR="pdf-output"
mkdir -p "$OUTPUT_DIR"

# Convert all numbered architecture docs
for MD_FILE in [0-9][0-9]_*.md; do
  if [ -f "$MD_FILE" ]; then
    OUTPUT_PDF="$OUTPUT_DIR/${MD_FILE%.md}.pdf"

    echo "Converting: $MD_FILE → $OUTPUT_PDF"

    pandoc "$MD_FILE" -o "$OUTPUT_PDF" \
      --pdf-engine=xelatex \
      --variable mainfont="DejaVu Sans Mono" \
      --variable geometry:margin=1in \
      --toc \
      --toc-depth=3 \
      --number-sections \
      --metadata title="TOSCA Architecture Documentation" \
      --metadata date="$(date +%Y-%m-%d)"
  fi
done

echo "All PDFs generated in $OUTPUT_DIR/"
```

Usage:

```bash
cd /docs/architecture
./convert-all-to-pdf.sh
```

Best Practices Summary

1. Image-Based Diagrams (Recommended)

- Replace ASCII diagrams with PlantUML-generated PNG/SVG images
- Keep PlantUML sources (`.puml` files) version-controlled
- Regenerate images when architecture changes
- Use PNG for PDF embedding (better compatibility)

2. Font Configuration

- Use XeLaTeX engine (best Unicode support)
- Specify monospace font: DejaVu Sans Mono, Courier New, Consolas
- Set appropriate font size (10-12pt for readability)

3. PDF Generation Options

- **Option A (Best Quality):** XeLaTeX with embedded images
- **Option B (Web-based):** wkhtmltopdf with CSS styling
- **Option C (Two-Pass):** Markdown → HTML → PDF

4. Dual-Format Maintenance

- Keep ASCII diagrams in markdown (quick viewing in Git, GitHub, text editors)
- Use conditional rendering (HTML comments) to show images in PDF
- Use pandoc filters to automate diagram replacement

5. Medical Device Compliance

- Professional-quality PDFs for regulatory submissions
- Version control diagram sources (traceability)

- Include metadata (title, date, version) in PDF headers/footers

---

Testing the Solution

Test Case 1: Single File with ASCII Diagram

```
# Before: Check current rendering
pandoc 03_safety_system.md -o test_before.pdf --pdf-engine=wkhtmltopdf

# After: Replace ASCII with image
# (Edit markdown to include image reference)
pandoc 03_safety_system.md -o test_after.pdf --pdf-engine=xelatex

# Compare output quality
```

Test Case 2: Verify Image Paths

```
# Ensure relative paths work from markdown location
cd /docs/architecture
ls -la diagrams/output/png/*.png

# Test markdown image reference
echo '![Test](diagrams/output/png/TOSCA%20System%20Context.png)' | \
  pandoc -o test_image.pdf --pdf-engine=xelatex
```

Test Case 3: Full Document with TOC

```
# Generate comprehensive PDF with table of contents
pandoc 01_system_overview.md -o 01_system_overview.pdf \
  --pdf-engine=xelatex \
  --toc \
  --toc-depth=3 \
  --number-sections \
  --metadata title="TOSCA System Overview" \
  --metadata date="2025-11-05"
```

Troubleshooting

Issue 1: Image Not Found Error

Symptom: `! LaTeX Error: File 'diagrams/output/png/diagram.png' not found.`

Solution:

```
# Check image path (relative to markdown file)
ls -la diagrams/output/png/

# Use absolute path if needed
DIAGRAM_DIR="/mnt/c/Users/wille/Desktop/TOSCA-dev/docs/architecture/diagrams/output/png"
pandoc 03_safety_system.md -o output.pdf \
  --pdf-engine=xelatex \
  --resource-path="$DIAGRAM_DIR"
```

Issue 2: Unicode Box Drawing Still Renders Poorly

Symptom: Box drawing characters appear as squares or question marks

Solution 1: Install Required Fonts

```
# Ubuntu/Debian
sudo apt install fonts-dejavu fonts-liberation

# macOS (via Homebrew)
```

```
brew tap homebrew/cask-fonts
brew install --cask font-dejavu-sans-mono

# Windows: Download DejaVu fonts from https://dejavu-fonts.github.io/
```

Solution 2: Force Font in LaTeX

```
pandoc file.md -o file.pdf \
  --pdf-engine=xelatex \
  --variable mainfont="DejaVu Sans Mono" \
  --variable monofont="DejaVu Sans Mono"
```

Issue 3: Images Too Large in PDF

Symptom: Images exceed page width or are too large

Solution: Add CSS/LaTeX Scaling

For wkhtmltopdf (CSS):

```
/* pdf-styles.css */
img {
  max-width: 80%;
  max-height: 500px;
  object-fit: contain;
}
```

For XeLaTeX (Markdown):

```
![Architecture Diagram](diagrams/output/png/diagram.png){width=80%}
```

Automation: CI/CD Integration

GitHub Actions Workflow

Create `.github/workflows/generate-docs-pdf.yml`:

```
name: Generate Architecture PDFs

on:
  push:
    paths:
      - 'docs/architecture/*.md'
      - 'docs/architecture/diagrams/*.puml'

jobs:
  generate-pdfs:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Repository
        uses: actions/checkout@v3

      - name: Install Dependencies
        run: |
          sudo apt update
          sudo apt install -y pandoc texlive-xetex fonts-dejavu

      - name: Generate PlantUML Diagrams
        run: |
          cd docs/architecture/diagrams
          java -jar plantuml.jar -tpng *.puml -o output/png

      - name: Convert Markdown to PDF
        run: |
          cd docs/architecture
          for md in [0-9][0-9]_*.md; do
            pandoc "$md" -o "pdf-output/${md%.md}.pdf" \
              --pdf-engine=xelatex \
              --variable mainfont="DejaVu Sans Mono" \
              --toc \
              --number-sections
          done

      - name: Upload PDF Artifacts
        uses: actions/upload-artifact@v3
        with:
          name: architecture-pdfs
          path: docs/architecture/pdf-output/*.pdf
```

Recommended Action Plan

Phase 1: Quick Fix (1-2 hours)

1. Use XeLaTeX engine instead of wkhtmltopdf for immediate improvement
2. Test with one sample file (e.g., `03_safety_system.md`)
3. Verify image paths and rendering quality

Phase 2: Image Integration (2-4 hours)

1. Map existing PlantUML images to markdown sections
2. Update markdown files to include image references
3. Use conditional rendering (HTML comments) to preserve ASCII for markdown viewing

Phase 3: Create Missing Diagrams (4-8 hours)

1. Identify diagrams not yet available as PlantUML
2. Create `.puml` sources for missing diagrams
3. Generate PNG/SVG outputs
4. Update markdown files

Phase 4: Automation (2-4 hours)

1. Create batch conversion script (`convert-all-to-pdf.sh`)
2. Set up CI/CD workflow for automatic PDF generation
3. Document workflow in README

Conclusion

Recommended Approach: 1. **Immediate:** Switch to XeLaTeX engine for better Unicode rendering 2. **Short-term:** Replace ASCII diagrams with PlantUML-generated images 3. **Long-term:** Automate diagram generation and PDF conversion in CI/CD

Key Benefits: - Professional-quality PDFs for regulatory submissions - Maintainable diagram-as-code workflow - Dual-format support (ASCII for quick markdown viewing, images for PDF) - Version-controlled architecture documentation

---

Document Version: 1.0 **Last Updated:** 2025-11-05 **Author:** Documentation Team