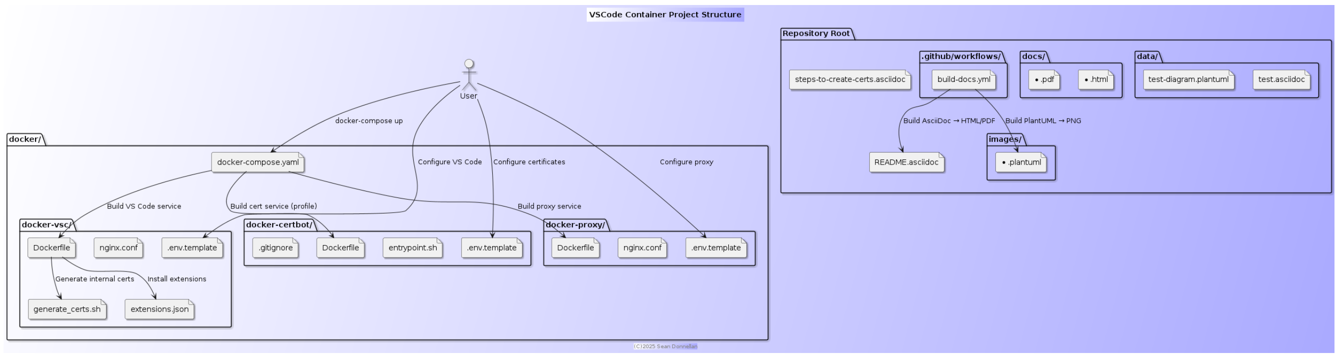
VSCode Container Project

A production-ready containerized VS Code server environment with SSL certificate support and reverse proxy architecture for secure remote development.



Service Discovery and Networking

The system uses CoreDNS for internal service discovery within the `vsagcrd.org` domain:

- **proxy.vsagcrd.org** (172.20.0.11) - Nginx reverse proxy with SSL termination
- **vscode.vsagcrd.org** (172.20.0.12) - VS Code Server application
- **coredns.vsagcrd.org** (172.20.0.10) - Internal DNS resolver
- **certbot.vsagcrd.org** (172.20.0.13) - Certificate management (run-once)

This architecture provides:

- Consistent internal hostname resolution
- Simplified container communication
- Production-ready network segregation
- Automatic service discovery

External Access

All external access flows through the proxy at `your-domain.vsagcrd.org:8443`:

```
Intranet → your-domain.vsagcrd.org:8443 → proxy.vsagcrd.org → vscode.vsagcrd.org
```

Architectural Decisions

Proxy-First Architecture

This project exclusively uses a reverse proxy architecture for the following reasons:

DNS Simplicity

- No need to modify local DNS or hosts files
- Works seamlessly with public domains
- Avoids port-specific access patterns that confuse users

Production Readiness

- Standard HTTPS on port 443 (not custom ports like 8443)
- Proper SSL termination at the edge
- HTTP to HTTPS automatic redirection

Security Benefits

- SSL/TLS handled by dedicated proxy layer
- Security headers automatically applied
- No direct container exposure to public internet

Operational Simplicity

- Single entry point for all traffic
- Centralized certificate management
- Standard web server patterns that ops teams understand

Future Extensibility

- Load balancing capabilities built-in
- Multiple backend support ready
- Monitoring and logging integration points

DNS Architecture: CoreDNS with Dynamic Local Detection

The project uses CoreDNS for internal service discovery with intelligent local DNS detection:

Local Network Awareness

- Automatically detects and uses local DNS servers (Fritz Box when home, mobile DNS when traveling)
- No hardcoded DNS servers (avoiding Google DNS, respecting privacy)
- Seamless portability across different network environments

Internal Service Discovery

- Simple `/etc/hosts` format for internal services (.vsagcrd.org domain)
- Automatic forward and reverse DNS generation

- Easy maintenance with single hosts file

Network Isolation Benefits

- Internal container communication via DNS names, not IP addresses
- Clear separation between internal services and external DNS
- Reduced configuration complexity for users

Quick Start

1. Clone this repository:

```
git clone https://github.com/donnels/vsc-container.git
cd vsc-container/docker
```

2. Generate Let's Encrypt certificates:

```
cp docker-certbot/.env.template docker-certbot/.env
# Edit docker-certbot/.env with your Cloudflare credentials and vsagcrd.org domain
docker-compose --profile tools run cert-generator
```

3. Configure proxy environment:

```
cp docker-proxy/.env.template docker-proxy/.env
# Edit docker-proxy/.env with your vsagcrd.org subdomain
```

4. Start the development environment:

```
docker-compose --profile dev up -d
```

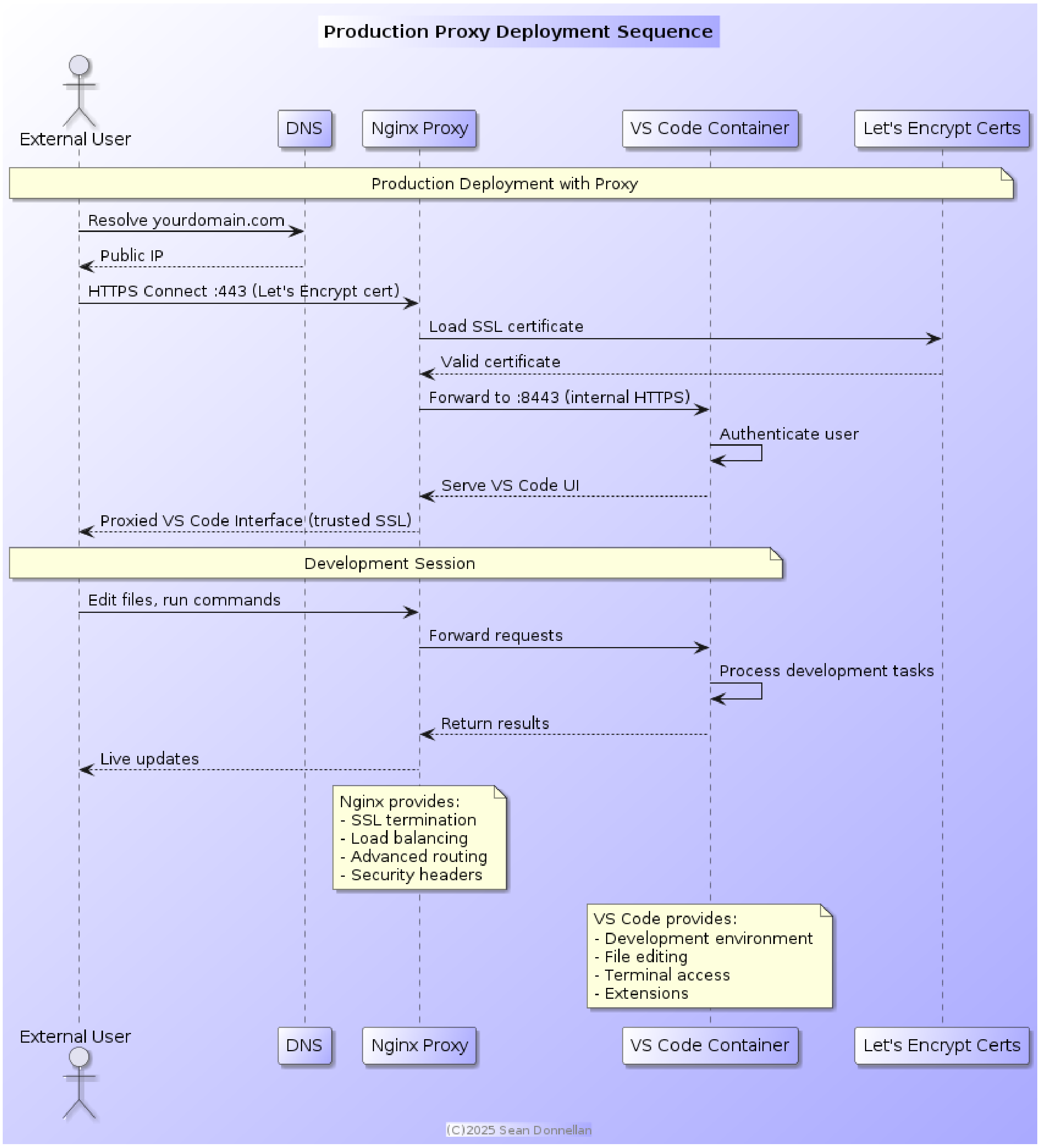
5. Access VS Code at: `https://your-subdomain.vsagcrd.org:8443`
6. Start the complete environment:

```
docker-compose up
```

7. Access VS Code at `https://vsagcrd.org`

Architecture Overview

The system uses a three-tier architecture designed for production deployment:


Production Proxy Deployment Sequence
External User
DNS
Nginx Proxy
VS Code Container
Let's Encrypt Certs
Production Deployment with Proxy
Resolve yourdomain.com
Public IP
HTTPS Connect :443 (Let's Encrypt cert)
Load SSL certificate
Valid certificate
Forward to :8443 (internal HTTPS)
Authenticate user
Serve VS Code UI
Proxied VS Code Interface (trusted SSL)
Development Session
Edit files, run commands
Forward requests
Process development tasks
Return results
Live updates
Nginx provides: - SSL termination - Load balancing - Advanced routing - Security headers
VS Code provides: - Development environment - File editing - Terminal access - Extensions
(C)2025 Sean Donnellan

Service Components

CoreDNS Service (`docker-coredns/`)

- Internal DNS resolution for `.vsagcrd.org` domain
- Service discovery for container-to-container communication
- Static IP assignment and hostname resolution
- Forwards external queries to public DNS servers

Nginx Reverse Proxy (`docker-proxy/`)

- SSL termination with Let's Encrypt certificates for `vsagcrd.org`

- HTTP to HTTPS redirection
- Security headers and modern TLS configuration
- WebSocket support for VS Code real-time features
- Standard ports: 80 (HTTP redirect) and 443 (HTTPS)

VS Code Service (`docker-vsc/`)

- Code-server accessible as `vscode.vsagcrd.org` internally
- Pre-installed development extensions (PlantUML, AsciiDoctor)
- Internal network access only (security isolation)
- Development tools: Java, Graphviz, mkcert, standard utilities

Certificate Generator (`docker-certbot/`)

- Let's Encrypt certificate generation for `vsagcrd.org` via Cloudflare DNS-01 challenge
- Automated certificate renewal capabilities
- On-demand execution via Docker Compose profiles
- Accessible as `certbot.vsagcrd.org` internally

Network Architecture

```
Internet → Nginx Proxy (443/80) → CoreDNS (172.20.0.10) → VS Code Container
(vscode.vsagcrd.org:8443)
                ▯                            ▯
          Let's Encrypt Certs      Internal DNS Resolution
              (Volume)              (.vsagcrd.org domain)
```

Service Discovery

- CoreDNS provides internal DNS resolution for `.vsagcrd.org` domain
- Each service has a static IP and hostname (e.g., `vscode.vsagcrd.org`)
- Container-to-container communication uses domain names, not IP addresses
- External DNS queries forwarded to public DNS servers (8.8.8.8, 1.1.1.1)

Security Boundaries

- Public internet only reaches the proxy service
- All internal services isolated on private network (172.20.0.0/16)
- Certificate volume shared read-only with proxy
- DNS resolution controlled by internal CoreDNS server

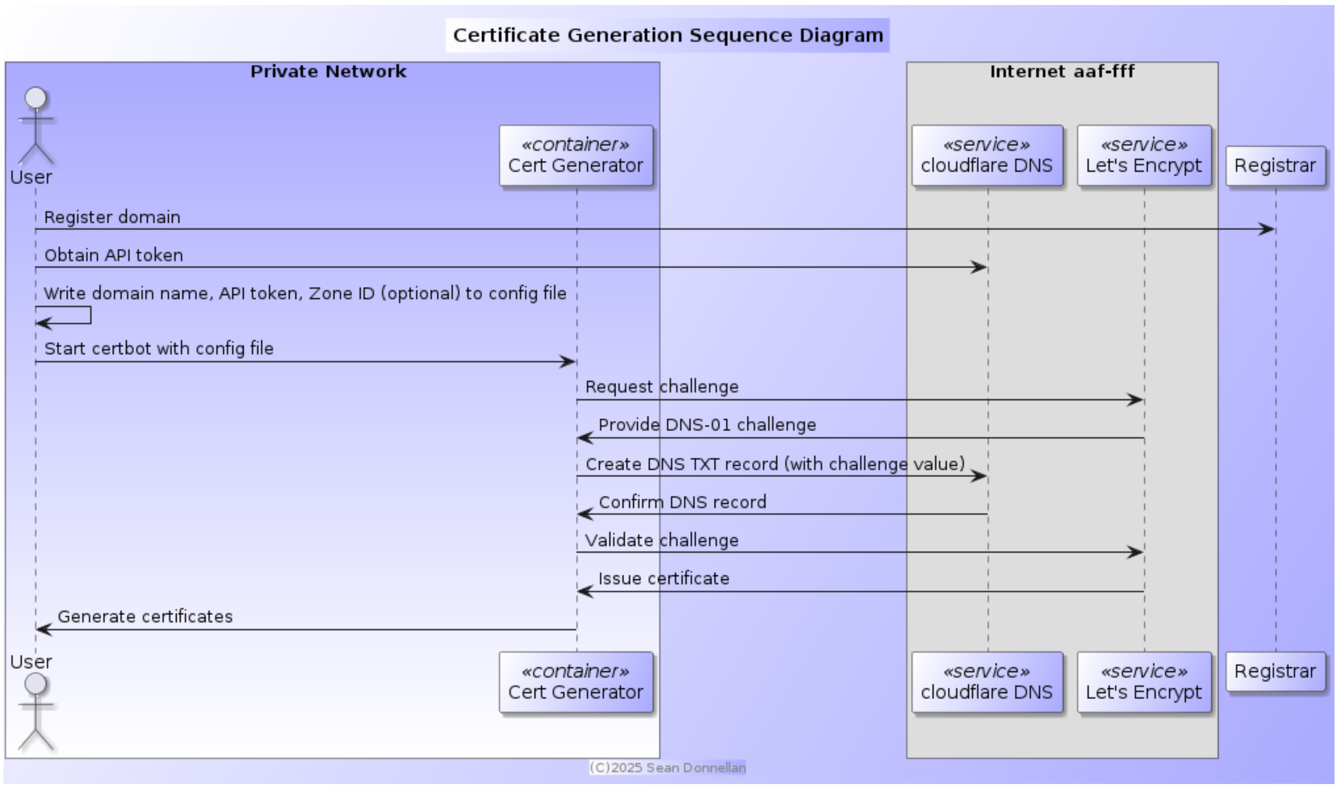
Directory Structure

```
docker/
├── docker-compose.yaml           # Complete orchestration
```

```
 ├─── docker-coredns/               # Internal DNS server
 │    ├─── Corefile                 # CoreDNS configuration
 │    ├─── vsagcrd.org.db          # Internal domain DNS records
 │    └─── reverse.db              # Reverse DNS lookup
 ├─── docker-proxy/                # Nginx reverse proxy
 │    ├─── Dockerfile
 │    ├─── nginx.conf               # Production SSL config
 │    └─── .env.template
 ├─── docker-vsc/                  # VS Code server
 │    ├─── Dockerfile
 │    ├─── generate_certs.sh        # Internal cert generation
 │    ├─── extensions.json          # Pre-installed extensions
 │    └─── .env.template
 └─── docker-certbot/              # Certificate management
     ├─── Dockerfile
     ├─── entrypoint.sh            # Let's Encrypt automation
     └─── .env.template
images/                           # Technical diagrams → PNG
docs/                            # Generated documentation
.github/workflows/               # Documentation automation
```

SSL Certificate Management

The system exclusively uses Let’s Encrypt certificates for production-grade SSL:



Certificate Generation Process

Prerequisites

1. Register your domain with a registrar
2. Configure domain DNS through Cloudflare
3. Create Cloudflare API token with DNS edit permissions: https://dash.cloudflare.com/profile/api-tokens
4. Obtain Cloudflare Zone ID for your domain

Certificate Generation Steps

1. Configure certificate generator:

   ```
   cd docker
   cp docker-certbot/.env.template docker-certbot/.env
   ```

2. Edit `docker-certbot/.env` with your credentials:

   ```
   DOMAIN=yoursubdomain.vsagcrd.org
   DOMAINS=yoursubdomain.vsagcrd.org,*.vsagcrd.org
   CLOUDFLARE_API_TOKEN=your_api_token_here
   CLOUDFLARE_ZONE_ID=your_zone_id_here
   ```

3. Generate certificates:

   ```
   docker-compose --profile tools run cert-generator
   ```

4. Certificates are automatically available to the proxy via Docker volume

Certificate Renewal

Certificates can be renewed by re-running the certificate generator:

```
docker-compose --profile tools run cert-generator
docker-compose restart proxy
```

For automated renewal, set up a cron job or systemd timer to run the generation command.

Development Environment

Included Development Tools

Pre-installed Extensions

- PlantUML: Technical diagram creation with Java runtime and Graphviz
- AsciiDoctor: Technical documentation authoring and preview

Development Stack

- VS Code Server with full extension support
- Git for version control
- Node.js development tools
- Java 17 runtime for PlantUML
- Standard Unix utilities (curl, wget, openssl)

Security Features

- Password-protected access
- HTTPS-only communication
- Internal network isolation
- Non-root container execution

Container Configuration

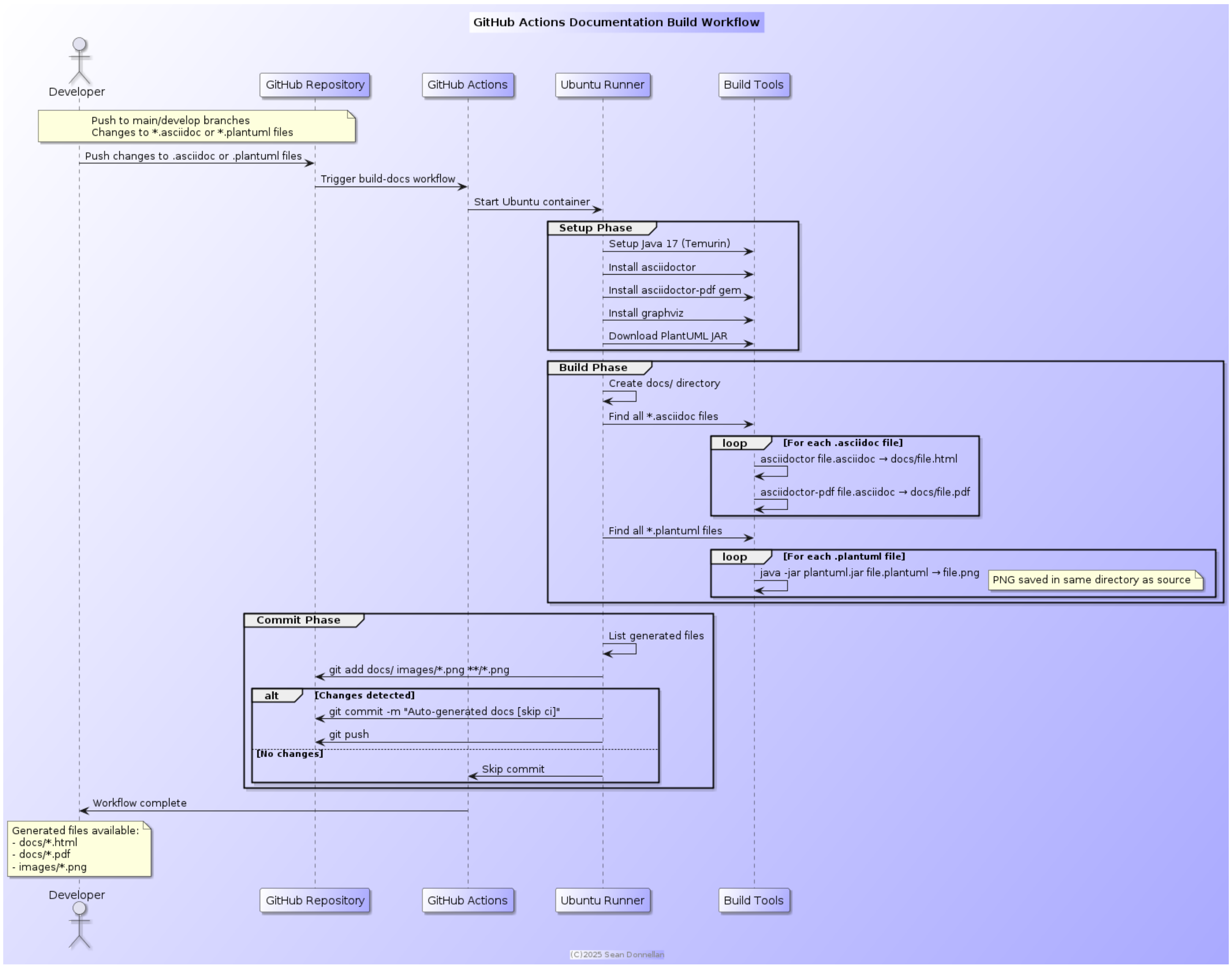
VS Code service configuration via environment variables:

```
# Certificate hostnames (for internal certificates)
CERT_HOSTNAMES=localhost 127.0.0.1

# Additional VS Code configuration can be added here
```

Documentation System

Automated documentation building powered by GitHub Actions:


GitHub Actions Documentation Build Workflow
Developer
GitHub Repository
GitHub Actions
Ubuntu Runner
Build Tools
Push to main/develop branches
Changes to *.asciidoc or *.plantuml files
Push changes to .asciidoc or .plantuml files
Trigger build-docs workflow
Start Ubuntu container
Setup Phase
Setup Java 17 (Temurin)
Install asciidoctor
Install asciidoctor-pdf gem
Install graphviz
Download PlantUML JAR
Build Phase
Create docs/ directory
Find all *.asciidoc files
loop
[For each .asciidoc file]
asciidoctor file.asciidoc → docs/file.html
asciidoctor-pdf file.asciidoc → docs/file.pdf
Find all *.plantuml files
loop
[For each .plantuml file]
java -jar plantuml.jar file.plantuml → file.png
PNG saved in same directory as source
Commit Phase
List generated files
git add docs/ images/*.png **/*.png
alt
[Changes detected]
git commit -m "Auto-generated docs [skip ci]"
git push
[No changes]
Skip commit
Workflow complete
Generated files available:
- docs/*.html
- docs/*.pdf
- images/*.png
(C)2025 Sean Donnellan

Documentation Architecture

Source Files

- `README.asciidoc`: Main project documentation (this file)
- `*.plantuml`: Technical diagrams and flowcharts
- `.adoc`, `.puml`: Include files for modular content

Generated Output

- HTML and PDF documentation in `docs/`
- PNG diagrams generated alongside source files
- Automated builds on every commit

Build Process

The GitHub Actions workflow automatically:

1. Triggers on changes to `.asciidoc` or `.plantuml` files
2. Installs AsciiDoctor, PlantUML, Java, and Graphviz
3. Converts documentation to HTML and PDF formats

4. Converts diagrams to PNG images
5. Commits generated files back to repository
6. Uses `[skip ci]` to prevent build loops

Deployment

Single Deployment Model

The system uses one standardized deployment approach:

1. **Certificate Generation**: Use Let's Encrypt with DNS validation
2. **Proxy Deployment**: Nginx handles SSL termination and routing
3. **Standard Ports**: HTTP (80) redirects to HTTPS (443)
4. **Domain-Based**: Requires proper domain configuration

Deployment Steps

1. Configure DNS to point your domain to the deployment server
2. Generate certificates using the certificate generator
3. Start all services with docker-compose
4. Access via standard HTTPS URL

Production Considerations

Domain Requirements

- Must have a registered domain name
- Domain must be configured in Cloudflare DNS
- DNS must resolve to the deployment server's public IP

Server Requirements

- Docker and Docker Compose installed
- Ports 80 and 443 accessible from internet
- Sufficient resources for development workloads

Security Considerations

- Automatic security headers via Nginx
- Modern TLS configuration (TLS 1.2+)
- HTTP Strict Transport Security (HSTS)
- No direct container exposure to internet

Operations and Maintenance

Monitoring and Logs

Access logs for troubleshooting:

```
# All services
docker-compose logs

# Specific services
docker-compose logs proxy
docker-compose logs code-server
docker-compose --profile tools logs cert-generator
```

Certificate Management

Monitor certificate expiration:

```
# Check certificate details
openssl x509 -in <(docker-compose exec proxy cat /etc/nginx/certs/fullchain.pem) -text
-noout
```

Automate renewal with cron:

```
# Add to crontab for monthly renewal
0 0 1 * * cd /path/to/project/docker && docker-compose --profile tools run cert-
generator && docker-compose restart proxy
```

Troubleshooting

Common Issues

- **Certificate Generation Fails**: Verify Cloudflare API token permissions and DNS configuration
- **Proxy Won't Start**: Check certificate files exist in volume
- **Access Denied**: Verify domain DNS points to correct server
- **WebSocket Issues**: Ensure proxy configuration includes WebSocket support

Diagnostic Commands

```
# Test certificate generation
docker-compose --profile tools run cert-generator

# Verify proxy configuration
```

```
docker-compose exec proxy nginx -t

# Check internal connectivity
docker-compose exec proxy curl -k https://code-server:8443
```

Extension and Customization

Adding Custom Extensions

Modify `docker-vsc/extensions.json` and rebuild:

```
{
  "recommendations": [
    "jebbs.plantuml",
    "asciidoctor.asciidoctor-vscode",
    "your.custom.extension"
  ]
}
```

Custom Nginx Configuration

Edit `docker-proxy/nginx.conf` for:

- Custom security headers
- Additional upstream services
- Rate limiting
- Custom routing rules

Documentation Extensions

- Add `.asciidoc` files anywhere for automatic building
- Create `.plantuml` files for automatic diagram generation
- Extend GitHub Actions workflow for additional build steps

Contributing

1. Fork the repository
2. Create feature branch: `git checkout -b feature/enhancement`
3. Test changes with full deployment
4. Ensure documentation builds correctly
5. Submit pull request with architectural justification

All documentation and diagrams are automatically built and updated via GitHub Actions.

Appendix A: Glossary

AsciiDoctor
: A fast text processor and publishing toolchain for converting AsciiDoc content to HTML5, DocBook, PDF, and other formats. Used in this project for documentation generation and technical writing with live preview capabilities.

Cloudflare
: A content delivery network and DNS service provider that offers domain management, security, and performance services. Used in this project for DNS management and API-based DNS-01 challenge validation for Let's Encrypt certificates.

CoreDNS
: A DNS server written in Go that chains plugins to provide flexible DNS services. Used in this project for internal service discovery within the vsagcrd.org domain, providing hostname resolution for container-to-container communication.

DNS-01 Challenge
: A Let's Encrypt ACME challenge method that validates domain ownership by requiring the client to create a specific DNS TXT record. Used with Cloudflare API for automated certificate generation without requiring the server to be publicly accessible.

Let's Encrypt
: A free, automated certificate authority that provides SSL/TLS certificates through the ACME protocol. Used in this project to generate production-grade SSL certificates for secure HTTPS communication.

PlantUML
: A tool for creating UML diagrams from plain text descriptions. Used in this project for generating technical architecture diagrams, sequence diagrams, and system documentation with Java runtime and Graphviz support.

Reverse Proxy
: A server that sits between clients and backend servers, forwarding client requests to backend servers and returning responses back to clients. In this project, Nginx serves as a reverse proxy providing SSL termination, security headers, and routing to the VS Code service.

SSL Termination
: The process of decrypting SSL/TLS traffic at a proxy server before forwarding unencrypted traffic to backend services. Implemented in this project at the Nginx proxy layer for centralized certificate management and security.

Vsagcrd
: Virtual Space and Global Communications Research Department - Place holder company created By Sean Donnelan as a hanger for projects like this one.

code-server

An open-source implementation of Visual Studio Code that runs on a remote server and is accessible through a web browser. Provides the full VS Code experience including extensions, integrated terminal, and debugging capabilities.

mkcert

A simple tool for making locally trusted development certificates. Used as the preferred method for generating internal SSL certificates for code-server when available, falling back to OpenSSL for self-signed certificates.