SOURCE: README.md

Legacy Application Modernization Platform - Setup

? Quick Start

After reinstalling Docker Desktop, you have **two deployment options**:

Option 1: PowerShell Script (Recommended)

```
# Full deployment with all services and AI models
.\rebuild-platform.ps1

# Skip AI model downloads (if already downloaded)
.\rebuild-platform.ps1 -SkipModels

# Skip monitoring stack
.\rebuild-platform.ps1 -SkipMonitoring

# Validation only
.\rebuild-platform.ps1 -Validate
```

Option 2: Docker Compose

```
# Start all services
docker-compose up -d

# View logs
docker-compose logs -f

# Stop all services
docker-compose down

# Stop and remove volumes (CAUTION: data loss)
docker-compose down -v
```

? --

? Prerequisites

? **Docker Desktop** installed and running
? **Windows 11 Pro** with WSL2 enabled
? **GPU Support** configured (for Ollama)
? **PowerShell 5.1+** or PowerShell Core 7+

? **Minimum 32GB RAM** (64GB+ recommended)
? **500GB+ available disk space** (for models and data)

? --

?? Platform Architecture

Infrastructure Services

Service	Port	Purpose	Credentials
PostgreSQL	5432	Relational database	‘modernization‘ / ‘modernization1
Redis	6379	Caching layer	No authentication
MinIO	9000, 9001	Object storage	‘minioadmin‘ / ‘minioadmin123‘
ChromaDB	8000	Vector database	No authentication
Ollama	11434	AI model server	No authentication

Monitoring Stack

Service	Port	Purpose	Credentials
Prometheus	9090	Metrics collection	No authentication
Grafana	3000	Visualization	‘admin‘ / ‘admin123‘
Loki	3100	Log aggregation	No authentication
Jaeger	16686	Distributed tracing	No authentication

AI Models (Ollama)

? **DeepSeek Coder 33B** (‘deepseek-coder:33b-instruct-q4_K_M‘) - ~18GB
? **Qwen2.5-Coder 32B** (‘qwen2.5-coder:32b-instruct-q4_K_M‘) - ~18GB

? --

? Validation & Health Checks

Quick Health Check

```
.\validate-platform.ps1
```

Detailed Health Check

```
.\validate-platform.ps1 -Detailed
```

Continuous Monitoring

```
# Check every 30 seconds
.\validate-platform.ps1 -Continuous -Interval 30
```

? --

?? Manual AI Model Management

Pull Additional Models

```
# List available models
docker exec ollama ollama list

# Pull a specific model
docker exec ollama ollama pull deepseek-coder:6.7b-instruct-q4_K_M

# Pull larger models (70B+)
docker exec ollama ollama pull qwen2.5-coder:72b-instruct-q4_K_M
```

Test Model Inference

```
# Test DeepSeek Coder
docker exec ollama ollama run deepseek-coder:33b-instruct-q4_K_M "Write a Python fu

# Test Qwen2.5-Coder
docker exec ollama ollama run qwen2.5-coder:32b-instruct-q4_K_M "Explain what a mic
```

? --

? Accessing Services

Web Interfaces

? **MinIO Console**: http://localhost:9001
? **Prometheus**: http://localhost:9090
? **Grafana**: http://localhost:3000
? **Jaeger**: http://localhost:16686

API Endpoints

? **Ollama API**: http://localhost:11434
? **ChromaDB API**: http://localhost:8000
? **Loki API**: http://localhost:3100

Database Connections

? *PostgreSQL**:

```
# Via Docker
docker exec -it postgres psql -U modernization -d legacy_modernization

# Connection string
postgresql://modernization:modernization123@localhost:5432/legacy_modernization
```

? *Redis**:

```
# Via Docker
docker exec -it redis redis-cli

# Connection string
redis://localhost:6379
```

? --

? Troubleshooting

Check Container Status

```
# View all containers
docker ps -a

# View specific container logs
docker logs -f <container-name>

# Check container resource usage
docker stats
```

Common Issues

1. Port Already in Use

```
# Find process using port
netstat -ano | findstr :<port>

# Kill process
Stop-Process -Id <PID> -Force
```

2. Container Won't Start

```
# Check logs
docker logs <container-name>

# Restart container
docker restart <container-name>

# Remove and recreate
docker rm -f <container-name>
.\rebuild-platform.ps1
```

3. Ollama GPU Not Detected

```
# Verify GPU support in Docker
docker run --rm --gpus all nvidia/cuda:12.2.0-base-ubuntu22.04 nvidia-smi

# Check Ollama container GPU access
docker exec ollama nvidia-smi
```

4. Out of Memory

```
# Check memory usage
docker stats --no-stream

# Restart Docker Desktop
# Or reduce number of running models
```

? --

?? Cleanup & Reset

Stop All Containers

```
docker stop $(docker ps -aq)
```

Remove All Containers

```
docker rm $(docker ps -aq)
```

Remove Volumes (?? DATA LOSS)

```
docker volume rm postgres-data redis-data minio-data chromadb-data ollama-data prom
```

Complete Reset

```
# Stop and remove everything
docker stop $(docker ps -aq)
```

```
docker rm $(docker ps -aq)
docker volume prune -f
docker network prune -f

# Rebuild from scratch
.\rebuild-platform.ps1
```

? --

? Monitoring & Metrics

Prometheus Targets

Visit http://localhost:9090/targets to verify all scrape targets are UP.

Grafana Dashboards

1. Log in to Grafana: http://localhost:3000 (admin / admin123) 2. Add Prometheus data source: - URL: ‘http://prometheus:9090‘ 3. Add Loki data source: - URL: ‘http://loki:3100‘ 4. Import dashboards for: - Ollama metrics - System metrics - Container metrics - GPU metrics

Common Queries

? *Prometheus**:

```
# Ollama request rate
rate(ollama_request_total[5m])

# Container CPU usage
rate(container_cpu_usage_seconds_total[5m])

# Memory usage
container_memory_usage_bytes / 1024 / 1024 / 1024
```

? --

? Security Considerations

Default Credentials (Change in Production!)

? PostgreSQL: ‘modernization‘ / ‘modernization123‘
? MinIO: ‘minioadmin‘ / ‘minioadmin123‘
? Grafana: ‘admin‘ / ‘admin123‘

Network Isolation

? Services are isolated in Docker networks
? Only necessary ports are exposed to host
? Consider using reverse proxy (Traefik/Nginx) for production

Data Persistence

? All data stored in Docker volumes
? Volumes persist after container removal
? Regular backup recommended for production

? --

? Next Steps

1. **Test Core Workflows**: - Legacy code analysis - Architecture diagram generation - Jira ticket generation

2. **Configure Monitoring**: - Set up Grafana dashboards - Configure alert rules in Prometheus - Set up log forwarding to Loki

3. **Deploy Application Layer**: - REST API services - Web UI components - Integration services

4. **Scale to Cloud**: - Plan Kubernetes migration - Set up CI/CD pipelines - Configure auto-scaling

? --

? Support & Resources

Platform Status

```
.\validate-platform.ps1 -Detailed
```

Docker Desktop Settings

? **WSL Integration**: Enabled
? **GPU Support**: Enabled
? **Memory**: 64GB+ allocated
? **Disk**: 500GB+ available

Useful Links

? [Docker Documentation](https://docs.docker.com/)
? [Ollama Documentation](https://github.com/ollama/ollama)
? [Prometheus Documentation](https://prometheus.io/docs/)
? [Grafana Documentation](https://grafana.com/docs/)

? --

? License & Attribution

? --

? *Last Updated**: 2026-01-16
? *Platform Version**: 1.0.0
? *Deployment Environment**: Windows 11 Pro + Docker Desktop + WSL2

SOURCE: INSTALL.md

INSTALLATION INSTRUCTIONS

? Download Complete!

You have downloaded the **Legacy Application Modernization Platform** deployment package.

? --

? Package Contents

This ZIP file contains: - ‘rebuild-platform.ps1‘ - Main deployment script - ‘validate-platform.ps1‘ - Health check and validation script - ‘docker-compose.yml‘ - Docker Compose configuration - ‘prometheus.yml‘ - Prometheus monitoring configuration - ‘loki-config.yml‘ -

Loki log aggregation configuration - ‘promtail-config.yml‘ - Promtail log collector configuration - ‘README.md‘ - Complete documentation - ‘QUICK-REFERENCE.md‘ - Command refer guide - ‘TROUBLESHOOTING.md‘ - Troubleshooting guide

? --

? Quick Start (3 Steps)

Step 1: Extract Files

Extract the ZIP file to a folder on your desktop, for example: ‘‘‘ C:\LegacyModernizationPlatform\ ‘‘‘

Step 2: Open PowerShell

Right-click on the folder and select **"Open in Terminal"** or: ‘‘‘powershell # Navigate to the folder cd C:\LegacyModernizationPlatform ‘‘‘

Step 3: Run Deployment

```
# Deploy everything
.\rebuild-platform.ps1
```

That’s it! The script will: ? Deploy all infrastructure services ? Deploy monitoring stack ? Download AI models ? Validate everything

? --

?? Expected Deployment Time

? **Infrastructure Services**: 2-3 minutes
? **Monitoring Stack**: 1-2 minutes
? **AI Models Download**: 15-30 minutes (one-time)
? **Total**: ~20-35 minutes for first run

Subsequent runs (with models already downloaded): ‘‘‘powershell .\rebuild-platform.ps1 -SkipModels ‘‘‘ Takes only: ~3-5 minutes

? --

? Prerequisites Check

Before running, ensure: - ? Docker Desktop is installed and running - ? WSL2 is enabled - ? GPU support is configured (for Ollama) - ? At least 32GB RAM available - ? At least 100GB free disk space - ? PowerShell 5.1+ or PowerShell Core 7+

Check Docker: ‘‘‘powershell docker --version docker info ‘‘‘

? --

? Script Parameters

```
# Full deployment (first time)
.\rebuild-platform.ps1

# Skip AI model downloads (if already downloaded)
.\rebuild-platform.ps1 -SkipModels

# Skip monitoring stack (minimal deployment)
.\rebuild-platform.ps1 -SkipMonitoring

# Validation only (check if everything is running)
.\rebuild-platform.ps1 -Validate
```

? --

? Post-Deployment Validation

After deployment completes, validate everything: ‘‘‘powershell .\validate-platform.ps1 -Detailed ‘‘‘

This shows: - Service health status - Port availability - Container status - AI models installed - Connectivity tests

? --

? Access Your Platform

After deployment, access services at:

? *Infrastructure:**
? PostgreSQL: ‘localhost:5432‘
? Redis: ‘localhost:6379‘
? MinIO Console: http://localhost:9001
? ChromaDB: http://localhost:8000
? Ollama: http://localhost:11434

? *Monitoring:**
? Prometheus: http://localhost:9090
? Grafana: http://localhost:3000 (admin/admin123)
? Loki: http://localhost:3100
? Jaeger: http://localhost:16686

? --

? Need Help?

1. **Check README.md** - Complete documentation 2. **Check QUICK-REFERENCE.md** - Commo
commands 3. **Check TROUBLESHOOTING.md** - Fix common issues

? *Common Issues:**

Port already in use: ‘‘‘powershell netstat -ano | findstr :<port> Stop-Process -Id <PID> -Force
‘‘‘

Container won’t start: ‘‘‘powershell docker logs <container-name> ‘‘‘

Complete reset: ‘‘‘powershell .\rebuild-platform.ps1 ‘‘‘

? --

? Verify Deployment Success

You should see output like: ‘‘‘ ? Platform Deployment Completed Successfully! ?

Infrastructure Services: ? PostgreSQL: localhost:5432 ? ? Redis: localhost:6379
? ? MinIO Console: http://localhost:9001 ? ? ChromaDB: http://localhost:8000 ? ?
Ollama: http://localhost:11434 ?

Monitoring Stack: ? Prometheus: http://localhost:9090 ? ? Grafana: http://localhost:3000 ? ? Loki: http://localhost:3100 ? ? Jaeger: http://localhost:16686 ? ‘‘‘

? --

? Support

If you encounter any issues:

1. Run validation: ‘‘‘powershell .\validate-platform.ps1 -Detailed ‘‘‘

2. Check container logs: ‘‘‘powershell docker logs <container-name> ‘‘‘

3. Review TROUBLESHOOTING.md for solutions

4. Generate diagnostic report: ‘‘‘powershell docker info > diagnostic-report.txt docker ps -a >> diagnostic-report.txt .\validate-platform.ps1 -Detailed >> diagnostic-report.txt ‘‘‘

? --

? Next Steps

After successful deployment:

1. **Test Core Services** ‘‘‘powershell # Test Ollama docker exec ollama ollama list

Test PostgreSQL docker exec postgres psql -U modernization -d legacy_modernization -c "SELECT version();"

Test Redis docker exec redis redis-cli PING ‘‘‘

2. **Configure Grafana Dashboards** - Visit http://localhost:3000 - Add Prometheus data source: http://prometheus:9090 - Add Loki data source: http://loki:3100

3. **Run Your Workflows** - Legacy code analysis - Architecture diagram generation - Jira ticket generation

? --

? Default Credentials

Remember to change these in production:

? **PostgreSQL**: modernization / modernization123
? **MinIO**: minioadmin / minioadmin123
? **Grafana**: admin / admin123

? --

? Enjoy Your Platform!

You now have a complete AI-powered Legacy Application Modernization Platform running locally with full monitoring and observability!

For detailed documentation, refer to README.md

? *Last Updated**: 2026-01-16
? *Platform Version**: 1.0.0

SOURCE: QUICK-REFERENCE.md

==========

QUICK REFERENCE CARD - Common Commands

==========

==========

? DEPLOYMENT COMMANDS

==========

Full platform rebuild

.\rebuild-platform.ps1

Rebuild without AI models

.\rebuild-platform.ps1 -SkipModels

Rebuild without monitoring

.\rebuild-platform.ps1 -SkipMonitoring

Docker Compose deployment

docker-compose up -d

Docker Compose with rebuild

docker-compose up -d --build --force-recreate

==========

? VALIDATION & HEALTH CHECKS

==========

Quick health check

.\validate-platform.ps1

Detailed health check with connectivity tests

.\validate-platform.ps1 -Detailed

Continuous monitoring (30 second intervals)

.\validate-platform.ps1 -Continuous -Interval 30

Check all container status

docker ps -a

Check specific service

docker ps --filter "name=ollama"

==========

? MONITORING COMMANDS

==========

View container logs (live)

docker logs -f <container-name>

View last 100 lines

docker logs --tail 100 <container-name>

View logs with timestamps

docker logs -t <container-name>

Resource usage (real-time)

docker stats

Resource usage (snapshot)

docker stats --no-stream

Container inspection

docker inspect <container-name>

==========

? OLLAMA AI MODEL COMMANDS

==========

List installed models

docker exec ollama ollama list

Pull new model

docker exec ollama ollama pull <model-name>

Examples:

docker exec ollama ollama pull deepseek-coder:6.7b-instruct-q4_K_M docker exec ollama ollama pull qwen2.5-coder:7b-instruct-q4_K_M docker exec ollama ollama pull codellama:13b-instruct-q4_K_M

Run model interactively

docker exec -it ollama ollama run deepseek-coder:33b-instruct-q4_K_M

Test model with prompt

docker exec ollama ollama run deepseek-coder:33b-instruct-q4_K_M "Write a Python function to calculate factorial"

Remove model

docker exec ollama ollama rm <model-name>

Show model info

docker exec ollama ollama show deepseek-coder:33b-instruct-q4_K_M

==========

? DATABASE COMMANDS

==========

PostgreSQL

Connect to PostgreSQL

docker exec -it postgres psql -U modernization -d legacy_modernization

Run SQL query

docker exec postgres psql -U modernization -d legacy_modernization -c "SELECT version();"

Backup database

docker exec postgres pg_dump -U modernization legacy_modernization > backup.sql

Restore database

cat backup.sql | docker exec -i postgres psql -U modernization -d legacy_modernization

Redis

Connect to Redis

docker exec -it redis redis-cli

Get all keys

docker exec redis redis-cli KEYS '*'

Get specific key

docker exec redis redis-cli GET <key>

Flush all data (CAUTION!)

docker exec redis redis-cli FLUSHALL

==========

?? STORAGE COMMANDS

==========

MinIO

Access MinIO console: http://localhost:9001

Credentials: minioadmin / minioadmin123

ChromaDB

Access ChromaDB API: http://localhost:8000

Check health

curl http://localhost:8000/api/v1/heartbeat

==========

? CONTAINER MANAGEMENT

==========

Start all containers

docker start $(docker ps -aq)

Stop all containers

docker stop $(docker ps -aq)

Restart specific container

docker restart <container-name>

Remove container

docker rm <container-name>

Remove container (force)

docker rm -f <container-name>

Recreate container

docker rm -f <container-name> .\rebuild-platform.ps1

==========

? VOLUME MANAGEMENT

==========

List volumes

docker volume ls

Inspect volume

docker volume inspect <volume-name>

Remove unused volumes

docker volume prune -f

Remove specific volume

docker volume rm <volume-name>

Backup volume

docker run --rm -v <volume-name>:/data -v ${PWD}:/backup alpine tar czf /backup/volume-backup.tar.gz -C /data .

Restore volume

docker run --rm -v <volume-name>:/data -v ${PWD}:/backup alpine tar xzf /backup/volume-backup.tar.gz -C /data

==========

? NETWORK COMMANDS

==========

List networks

docker network ls

Inspect network

docker network inspect modernization-network

Connect container to network

docker network connect <network-name> <container-name>

Disconnect container from network

docker network disconnect <network-name> <container-name>

Remove network

docker network rm <network-name>

==========

? CLEANUP COMMANDS

==========

Stop all containers

docker stop $(docker ps -aq)

Remove all containers

docker rm $(docker ps -aq)

Remove unused images

docker image prune -a -f

Remove unused volumes

docker volume prune -f

Remove unused networks

docker network prune -f

Complete cleanup (CAUTION: removes everything)

docker system prune -a -f --volumes

==========

? TROUBLESHOOTING COMMANDS

==========

Check Docker Desktop status

docker info

Check WSL2 integration

wsl -l -v

Check GPU availability

docker run --rm --gpus all nvidia/cuda:12.2.0-base-ubuntu22.04 nvidia-smi

Check port usage

netstat -ano | findstr :<port>

Test connectivity to service

Test-NetConnection -ComputerName localhost -Port <port>

Check container IP address

docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' <container-name>

Execute command in container

docker exec <container-name> <command>

Get shell in container

docker exec -it <container-name> /bin/sh docker exec -it <container-name> /bin/bash

==========

? PROMETHEUS QUERIES

==========

Access Prometheus: http://localhost:9090

Example queries:

Ollama request rate

rate(ollama_request_total[5m])

Container CPU usage

rate(container_cpu_usage_seconds_total[5m]) * 100

Container memory usage (GB)

container_memory_usage_bytes / 1024 / 1024 / 1024

GPU utilization

nvidia_gpu_utilization

Disk usage

(node_filesystem_size_bytes - node_filesystem_free_bytes) / node_filesystem_size_bytes * 100

==========

? GRAFANA COMMANDS

==========

Access Grafana: http://localhost:3000

Default credentials: admin / admin123

Add Prometheus data source:

Configuration > Data Sources > Add data source >

URL: http://prometheus:9090

Add Loki data source:

Configuration > Data Sources > Add data source >

URL: http://loki:3100

==========

? SECURITY COMMANDS

==========

View container environment variables

```
docker exec <container-name> env
```

Check file permissions in container

```
docker exec <container-name> ls -la /path
```

Update password (example: Grafana)

```
docker exec grafana grafana-cli admin reset-admin-password <new-password>
```

==========

? LOGGING COMMANDS

==========

View logs from all containers

```
docker-compose logs -f
```

View logs from specific service

docker-compose logs -f <service-name>

Export logs to file

docker logs <container-name> > container-logs.txt 2>&1

Follow logs with grep filter

docker logs -f <container-name> | grep ERROR

==========

? EMERGENCY COMMANDS

==========

Complete platform restart

docker restart $(docker ps -aq)

Force stop all containers

docker kill $(docker ps -aq)

Rebuild everything from scratch

docker stop $(docker ps -aq) docker rm $(docker ps -aq) docker volume rm $(docker volume ls -q) docker network rm modernization-network monitoring-network .\rebuild-platform.ps1

Restart Docker Desktop

Stop-Service docker

Start-Service docker

Or restart via Docker Desktop UI

==========

? SUPPORT COMMANDS

==========

Generate system report

docker info > system-report.txt docker ps -a >> system-report.txt docker stats --no-stream >> system-report.txt docker volume ls >> system-report.txt docker network ls >> system-report.txt

Export configuration

docker-compose config > current-config.yml

Check version information

docker version docker-compose version

==========

END OF QUICK REFERENCE

==========

SOURCE: TROUBLESHOOTING.md

? Troubleshooting Guide

Common Issues and Solutions

? --

? Docker Desktop Issues

Issue: Docker Desktop Won’t Start

? *Symptoms:**
? Docker Desktop stuck at "Starting..."
? Error: "Docker Desktop is not running"

? *Solutions:**

1. **Restart Docker Desktop**: ‘‘‘powershell # Via PowerShell (as Administrator) Stop-Service docker Start-Service docker

Or restart via Docker Desktop UI ‘‘‘

2. **Check WSL2 Status**: ‘‘‘powershell wsl -l -v # Ensure WSL2 is running wsl --update ‘‘‘

3. **Reset Docker Desktop**: - Open Docker Desktop - Settings > Trouble shoot > Reset to factory defaults - **WARNING**: This will delete all containers, images, and volumes

4. **Check Windows Services**: - Open Services (services.msc) - Ensure "Docker Desktop Service" is running - Start type should be "Automatic"

? --

? Port Already in Use

Issue: Port Conflict When Starting Containers

? *Symptoms:**
? Error: "Bind for 0.0.0.0:5432 failed: port is already allocated"
? Container fails to start

? *Solutions:**

1. **Identify Process Using Port**: ‘‘‘powershell # Find process ID netstat -ano | findstr :<port>

Example for PostgreSQL (port 5432) netstat -ano | findstr :5432 ‘‘‘

2. **Kill Process**: ‘‘‘powershell # Kill process by PID Stop-Process -Id <PID> -Force

Example Stop-Process -Id 12345 -Force ‘‘‘

3. **Change Port in Configuration**: - Edit docker-compose.yml - Change port mapping (e.g., "5433:5432" instead of "5432:5432")

? --

? Container Issues

Issue: Container Keeps Restarting

? *Symptoms:**
? Container status shows "Restarting"
? Container exits immediately after start

? *Solutions:**

1. **Check Container Logs**: ‘‘‘powershell docker logs <container-name> ‘‘‘

2. **Inspect Container**: ‘‘‘powershell docker inspect <container-name> ‘‘‘

3. **Common Causes**: - **Configuration Error**: Check environment variables - **Port Conflict**: Ensure port is available - **Volume Mount Issue**: Verify volume paths exist - **Insufficient Resources**: Check RAM/CPU allocation

4. **Fix and Restart**: ‘‘‘powershell # Remove problematic container docker rm -f <container-name>

Rebuild .\rebuild-platform.ps1 ‘‘‘

Issue: Container is Unhealthy

? *Symptoms:**
? Docker ps shows "unhealthy" status
? Health check failing

? *Solutions:**

1. **Check Health Check Logs**: ‘‘‘powershell docker inspect --format=’{{json .State.Health}}’ <container-name> ‘‘‘

2. **Manual Health Check**: ‘‘‘powershell # Example for Ollama curl http://localhost:11434/

Example for ChromaDB curl http://localhost:8000/api/v1/heartbeat ‘‘‘

3. **Restart Container**: ‘‘‘powershell docker restart <container-name> ‘‘‘

? --

? Ollama Issues

Issue: GPU Not Detected by Ollama

? *Symptoms:**
? Ollama running on CPU only
? Slow model inference
? nvidia-smi not working in container

? *Solutions:**

1. **Verify Docker GPU Support**: ‘‘‘powershell docker run --rm --gpus all nvidia/cuda:12.2.0-base-ubuntu22.04 nvidia-smi ‘‘‘

2. **Check Ollama Container GPU Access**: ‘‘‘powershell docker exec ollama nvidia-smi ‘‘‘

3. **Verify Docker Desktop GPU Settings**: - Docker Desktop > Settings > Resources - Ensure "Use GPU" is enabled - Restart Docker Desktop

4. **Recreate Ollama Container**: ‘‘‘powershell docker rm -f ollama .\rebuild-platform.ps1 ‘‘‘

Issue: Model Download Fails

? *Symptoms:**
? Error: "pull model manifest"
? Download interrupted
? Insufficient disk space

? *Solutions:**

1. **Check Disk Space**: ‘‘‘powershell # Check available space Get-PSDrive C ‘‘‘

2. **Retry Download**: ‘‘‘powershell docker exec ollama ollama pull <model-name> ‘‘‘

3. **Download Smaller Model**: ‘‘‘powershell # Use 7B instead of 33B docker exec ollama ollama pull deepseek-coder:6.7b-instruct-q4_K_M ‘‘‘

4. **Clear Failed Downloads**: ‘‘‘powershell docker exec ollama ollama rm <model-name> ‘‘‘

Issue: Model Inference is Slow

? *Symptoms:**
? Long response times
? High CPU usage
? Out of memory errors

? *Solutions:**

1. **Check GPU Utilization**: ‘‘‘powershell nvidia-smi ‘‘‘

2. **Use Smaller Model or Lower Quantization**: - Q4_K_M (medium quality, faster) - Q2_K (lower quality, fastest)

3. **Reduce Concurrent Requests**: - Limit parallel inference calls - Implement request queuing

4. **Increase GPU Memory**: - Close other GPU-intensive applications - Use model with smaller parameters

? --

? Database Issues

Issue: PostgreSQL Connection Refused

? *Symptoms:**
? Error: "could not connect to server"
? Connection timeout

? *Solutions:**

1. **Verify Container is Running**: ‘‘‘powershell docker ps --filter "name=postgres" ‘‘‘

2. **Check Logs**: ‘‘‘powershell docker logs postgres ‘‘‘

3. **Test Connection**: ‘‘‘powershell docker exec postgres psql -U modernization -d

legacy_modernization -c "SELECT version();" ‘‘‘

4. **Verify Port is Open**: ‘‘‘powershell Test-NetConnection -ComputerName localhost -Port 5432 ‘‘‘

Issue: Redis Connection Issues

? *Symptoms:**
? Error: "Could not connect to Redis"
? Commands timing out

? *Solutions:**

1. **Test Redis**: ‘‘‘powershell docker exec redis redis-cli PING # Should return PONG ‘‘‘

2. **Check Redis Logs**: ‘‘‘powershell docker logs redis ‘‘‘

3. **Restart Redis**: ‘‘‘powershell docker restart redis ‘‘‘

? --

? Monitoring Issues

Issue: Prometheus Targets Down

? *Symptoms:**
? Targets showing as "DOWN" in Prometheus UI
? No metrics being collected

? *Solutions:**

1. **Check Target Accessibility**: ‘‘‘powershell # Test Ollama metrics curl http://localhost:11434/metrics ‘‘‘

2. **Verify Network Connectivity**: ‘‘‘powershell docker exec prometheus wget -O- http://ollama:11434/metrics ‘‘‘

3. **Check Prometheus Configuration**: ‘‘‘powershell docker exec prometheus cat /etc/prometheus/prometheus.yml ‘‘‘

4. **Reload Prometheus Configuration**: ‘‘‘powershell # Via API Invoke-WebRequest -Method POST

http://localhost:9090/-/reload

Or restart docker restart prometheus ‘‘‘

Issue: Grafana Dashboard Shows No Data

? *Symptoms:**
? Empty graphs
? "No data" message

? *Solutions:**

1. **Verify Data Source Connection**: - Grafana > Configuration > Data Sources - Test connection to Prometheus/Loki

2. **Check Query Syntax**: - Verify PromQL/LogQL queries - Test queries in Prometheus UI first

3. **Verify Time Range**: - Check dashboard time picker - Ensure data exists for selected time range

4. **Check Prometheus Scrape Status**: - Visit http://localhost:9090/targets - Verify all targets are UP

? --

?? Storage Issues

Issue: Volume Mount Fails

? *Symptoms:**
? Container can’t access volume
? Permission denied errors
? Data not persisting

? *Solutions:**

1. **Check Volume Exists**: ‘‘‘powershell docker volume ls | Select-String <volume-name> ‘‘‘

2. **Recreate Volume**: ‘‘‘powershell docker volume rm <volume-name> docker volume create <volume-name> ‘‘‘

3. **Check Volume Permissions**: ‘‘‘powershell docker exec <container-name> ls -la /path/to/mount ‘‘‘

Issue: Disk Space Full

? *Symptoms:**
? "No space left on device"
? Containers failing to start
? Unable to download models

? *Solutions:**

1. **Check Disk Usage**: ‘‘‘powershell docker system df ‘‘‘

2. **Clean Up Docker**: ‘‘‘powershell # Remove unused images docker image prune -a -f

Remove unused volumes (CAUTION) docker volume prune -f

Complete cleanup docker system prune -a -f --volumes ‘‘‘

3. **Move Docker Data Location**: - Docker Desktop > Settings > Resources > Advanced - Change "Disk image location"

? --

? Network Issues

Issue: Containers Can’t Communicate

? *Symptoms:**
? Services can’t reach each other
? Network timeout errors

? *Solutions:**

1. **Verify Network Exists**: ‘‘‘powershell docker network ls | Select-String modernization ‘‘‘

2. **Check Container Network Connections**: ‘‘‘powershell docker inspect <container-name> --format=’{{json .NetworkSettings.Networks}}’ ‘‘‘

3. **Reconnect Container to Network**: ‘‘‘powershell docker network connect modernization-

network <container-name> ‘‘‘

4. **Recreate Network**: ‘‘‘powershell docker network rm modernization-network docker network create modernization-network ‘‘‘

? --

? Authentication Issues

Issue: Can’t Connect to MinIO Console

? *Symptoms:**
? Login fails
? "Access Denied" error

? *Solutions:**

1. **Verify Credentials**: - Username: ‘minioadmin‘ - Password: ‘minioadmin123‘

2. **Check Environment Variables**: ‘‘‘powershell docker exec minio env | Select-String MINIO_ROOT ‘‘‘

3. **Reset MinIO**: ‘‘‘powershell docker rm -f minio docker volume rm minio-data .\rebuild-platform.ps1 ‘‘‘

Issue: Grafana Login Issues

? *Symptoms:**
? Default credentials not working
? Locked out of Grafana

? *Solutions:**

1. **Reset Admin Password**: ‘‘‘powershell docker exec grafana grafana-cli admin reset-admin-password newpassword123 ‘‘‘

2. **Check Environment Variables**: ‘‘‘powershell docker exec grafana env | Select-String GF_SECURITY ‘‘‘

? --

? Performance Issues

Issue: High CPU Usage

? *Symptoms:**
? System running slow
? Containers using 100% CPU

? *Solutions:**

1. **Identify Resource Hog**: ‘‘‘powershell docker stats --no-stream ‘‘‘

2. **Limit Container Resources**: ‘‘‘yaml # In docker-compose.yml services: ollama: deploy: resources: limits: cpus: ’8’ memory: 32G ‘‘‘

3. **Reduce Concurrent Operations**: - Limit parallel model inference - Reduce scrape frequency in Prometheus

Issue: High Memory Usage

? *Symptoms:**
? System running out of RAM
? OOM (Out of Memory) errors

? *Solutions:**

1. **Check Memory Usage**: ‘‘‘powershell docker stats --no-stream --format "table {{.Container}}\t{{.MemUsage}}" ‘‘‘

2. **Increase Docker Memory Allocation**: - Docker Desktop > Settings > Resources - Increase Memory slider

3. **Use Smaller Models**: - Switch to 7B models instead of 33B - Use higher quantization (Q2_K instead of Q4_K_M)

? --

? Configuration Issues

Issue: Changes Not Taking Effect

? *Symptoms:**

? Configuration updates ignored
? Old settings persist

? *Solutions:**

1. **Reload Configuration**: “‘powershell # For Prometheus Invoke-WebRequest -Method POST http://localhost:9090/-/reload “‘

2. **Recreate Containers**: “‘powershell docker-compose down docker-compose up -d --force-recreate “‘

3. **Clear Docker Cache**: “‘powershell docker-compose build --no-cache docker-compose up -d “‘

? --

? Emergency Recovery

Complete Platform Reset

If nothing else works, perform a complete reset:

```
# 1. Stop everything
docker stop $(docker ps -aq)

# 2. Remove all containers
docker rm $(docker ps -aq)

# 3. Remove all volumes (DATA LOSS!)
docker volume prune -f

# 4. Remove all networks
docker network prune -f

# 5. Remove all images (optional)
docker image prune -a -f

# 6. Restart Docker Desktop
# - Docker Desktop > Restart

# 7. Rebuild platform
.\rebuild-platform.ps1
```

? --

? Getting Help

Collect Diagnostic Information

```
# Generate system report
docker info > system-report.txt
docker ps -a >> system-report.txt
docker stats --no-stream >> system-report.txt
docker volume ls >> system-report.txt
docker network ls >> system-report.txt
.\validate-platform.ps1 -Detailed >> system-report.txt
```

Check Service Logs

```
# Export logs from all containers
docker logs postgres > logs/postgres.log 2>&1
docker logs redis > logs/redis.log 2>&1
docker logs ollama > logs/ollama.log 2>&1
docker logs prometheus > logs/prometheus.log 2>&1
```

? --

? Prevention Tips

1. **Regular Health Checks**: ‘‘‘powershell .\validate-platform.ps1 -Continuous -Interval 300 ‘‘‘

2. **Monitor Resource Usage**: - Keep an eye on CPU/RAM/Disk usage - Set up alerts in Prometheus

3. **Regular Backups**: - Backup volumes regularly - Export important configurations

4. **Keep Docker Updated**: - Update Docker Desktop regularly - Update container images periodically

5. **Documentation**: - Document custom configurations - Keep track of changes

? --

? *Last Updated**: 2026-01-16

SOURCE: RUNNER-SETUP.md

GitLab Runner with Docker-in-Docker (DinD) Setup

Complete guide for configuring a GitLab runner capable of building and pushing Docker images to Nexus registry.

Quick Start

```
cd files
.\setup-gitlab-runner-dind.ps1 -GitLabUrl "http://gitlab-server" -RunnerName "docke
```

The script will: 1. Verify Docker is running 2. Create the ‘modernization-network‘ bridge (if needed) 3. Generate runner configuration with DinD support 4. Optionally start the runner container

Architecture

```
GitLab CI/CD Pipeline (.gitlab-ci.yml)
        ?
  docker_build job
        ?
Runner (docker executor)
        ?
DinD Service (docker:24-dind)
        ?
Docker Socket Sharing (/var/run/docker.sock)
        ?
Build & Push to Nexus Registry
```

Configuration Details

Runner Configuration (‘runner-config.toml‘)

Key settings for Docker-in-Docker:

```
[[runners]]
  name = "docker-dind-runner"
  executor = "docker"
```

```
  [runners.docker]
    image = "docker:24-cli"           # Lightweight Docker client
    privileged = true                 # Required for Docker operations
    volumes = [
      "/cache",
      "/var/run/docker.sock:/var/run/docker.sock"  # Access host Docker daemon
    ]
    shm_size = 2147483648            # 2GB for builds
    network_mode = "modernization-network"

    [[runners.docker.services]]
      name = "docker:24-dind"         # Docker-in-Docker service
      alias = "docker"
      command = ["--tls=false"]       # Disable TLS for reliability
```

CI/CD Pipeline Configuration

In your ‘.gitlab-ci.yml‘:

```
docker_build:
  stage: docker
  image: docker:24-cli
  services:
    - name: docker:24-dind
      alias: docker
      command: ["--tls=false"]

  variables:
    DOCKER_TLS_CERTDIR: ""             # Required for DinD
    DOCKER_HOST: tcp://docker:2375     # DinD endpoint
    DOCKER_DRIVER: overlay2
    FF_NETWORK_PER_BUILD: "true"       # Per-build network isolation
    FF_DOCKER_LAYER_CACHING: "true"    # Cache Docker layers
    DOCKER_REGISTRY: nexus-docker:5001
    DOCKER_IMAGE_NAME: java-app

  before_script:
    - sleep 5                          # Wait for DinD service
    - docker info                      # Verify connectivity
    - docker login -u $DOCKER_REGISTRY_USER -p $DOCKER_REGISTRY_PASSWORD $DOCKER_RE

  script:
    - docker build -t $DOCKER_REGISTRY/$DOCKER_IMAGE_NAME:latest .
    - docker push $DOCKER_REGISTRY/$DOCKER_IMAGE_NAME:latest
```

Manual Runner Registration

If you prefer manual setup instead of the PowerShell script:

1. Create Configuration File

Create ‘~/.gitlab-runner/config.toml‘:

```
concurrent = 3
check_interval = 0

[session_server]
  session_timeout = 1800

[[runners]]
  name = "docker-dind-runner"
  url = "http://gitlab-server"
  token = "YOUR_REGISTRATION_TOKEN"
  executor = "docker"

  [runners.docker]
    image = "docker:24-cli"
    privileged = true
    volumes = ["/cache", "/var/run/docker.sock:/var/run/docker.sock"]
    shm_size = 2147483648
    network_mode = "modernization-network"

    [[runners.docker.services]]
      name = "docker:24-dind"
      alias = "docker"
      command = ["--tls=false"]
```

2. Run the Runner Container

```
docker run -d \
  --name gitlab-runner \
  --restart unless-stopped \
  --network modernization-network \
  -v ~/.gitlab-runner/config.toml:/etc/gitlab-runner/config.toml:ro \
  -v /var/run/docker.sock:/var/run/docker.sock \
  gitlab/gitlab-runner:latest
```

3. Verify the Runner is Running

```
docker logs -f gitlab-runner
```

GitLab CI/CD Variables Setup

Set these in GitLab (Settings > CI/CD > Variables):

Variable	Value	Masked
‘DOCKER_REGISTRY_USER‘	Nexus username	No
‘DOCKER_REGISTRY_PASSWORD‘	Nexus password	**Yes**
‘DOCKER_REGISTRY‘	‘nexus-docker:5001‘	No
‘DOCKER_IMAGE_NAME‘	‘java-app‘	No

Network Configuration

The runner must be on the same Docker network as Nexus:

```
# Verify network exists
docker network ls | grep modernization-network

# Create if missing
docker network create modernization-network
```

All services communicate by container name: - ‘nexus-docker:5001‘ ? Nexus Docker registry - ‘docker:2375‘ ? DinD service endpoint

Troubleshooting

Runner Won’t Connect to GitLab

```
# Check runner logs
docker logs gitlab-runner

# Verify GitLab is reachable
docker exec gitlab-runner ping gitlab-server

# Test from runner container
docker run --rm --network modernization-network curlimages/curl curl http://gitlab-
```

Docker Commands Fail in Pipeline

```
# Check DinD service connectivity
docker exec gitlab-runner docker info

# Verify socket binding
docker inspect gitlab-runner | grep -A 10 Mounts
```

Image Push Fails to Nexus

```
# Verify Nexus is accessible
docker run --rm --network modernization-network curlimages/curl curl http://nexus-d

# Check registry credentials in pipeline logs
# (ensure DOCKER_REGISTRY_PASSWORD is masked in CI/CD variables)

# Test login locally
docker login -u admin -p admin123 nexus-docker:5001
```

Out of Disk Space During Builds

The ‘shm_size = 2147483648‘ (2GB) might need adjustment: - Increase for large Docker image builds - Decrease if system RAM is limited

Update in ‘runner-config.toml‘ and restart runner:

```
docker restart gitlab-runner
```

Maintenance

View Active Jobs

```
docker exec gitlab-runner gitlab-runner --debug list
```

Update Runner Version

```
docker pull gitlab/gitlab-runner:latest
docker restart gitlab-runner
```

Rotate Runner Token

1. In GitLab: Settings > CI/CD > Runners > Click runner > Rotate token 2. Update ‘runner-config.toml‘ with new token 3. Restart runner: ‘docker restart gitlab-runner‘

Performance Tips

1. **Cache Docker Layers**: Enable ‘FF_DOCKER_LAYER_CACHING‘ to reuse build layers 2. **Parallel Builds**: Set ‘concurrent = 3‘ (or higher) for multiple simultaneous jobs 3. **Shared Memory**: Increase ‘shm_size‘ if experiencing OOM errors 4. **Network**: Use

‘network_mode = "modernization-network"‘ for service communication

Security Considerations

?? **Production Deployment**: - Use TLS with proper certificates (set ‘tls_verify = true‘) - Don’t share ‘/var/run/docker.sock‘ in production (security risk) - Use dedicated runner host with proper isolation - Rotate credentials regularly - Use masked CI/CD variables for secrets - Restrict runner to specific projects/tags

For production, consider using Kubernetes executor with isolated pod security policies instead of Docker executor.

References

? [GitLab Runner Documentation](https://docs.gitlab.com/runner/)
? [Docker-in-Docker Executor](https://docs.gitlab.com/runner/executors/docker.html#docker-in-docker)
? [Docker Service Variables](https://docs.gitlab.com/ee/ci/services/)
? [Nexus Docker Repository](https://help.sonatype.com/repomanager3/formats/docker-registry)

SOURCE: RUNNER-SETUP-SUMMARY.md

GitLab Runner DinD Configuration - Complete Setu

What Was Configured

Your GitLab runner is now configured with **Docker-in-Docker (DinD)** capability for building and pushing Docker images to Nexus registry.

Files Modified/Created

1. Runner Configuration

? *java-pipeline/runner-config.toml**
? Updated concurrent jobs from 1 to 3
? Added DinD service configuration with ‘docker:24-dind‘

? Enabled Docker socket sharing (‘/var/run/docker.sock‘)
? Configured ‘modernization-network‘ for service communication
? Set 2GB shared memory for Docker operations

2. CI/CD Pipeline

? *java-pipeline/.gitlab-ci.yml**
? Enhanced Docker build job with DinD variables
? Added environment variables for Docker registry
? Configured DinD service with TLS disabled (for reliability)
? Added layer caching flags for faster builds

3. Setup Script (NEW)

? *files/setup-gitlab-runner-dind.ps1**
? PowerShell script for automatic runner setup
? Creates runner configuration with DinD enabled
? Starts GitLab runner container
? Provides comprehensive setup guidance

4. Documentation (NEW)

? *files/RUNNER-SETUP.md**
? Complete runner setup guide
? Configuration explanations
? Troubleshooting section
? Security considerations
? Performance optimization tips

Quick Setup Steps

Step 1: Run the Setup Script

```
cd files
.\setup-gitlab-runner-dind.ps1 -GitLabUrl "http://gitlab-server" -RunnerName "docke
```

Step 2: Configure GitLab CI/CD Variables

In GitLab UI (Settings > CI/CD > Variables): - ‘DOCKER_REGISTRY_USER‘: Your Nexus username - ‘DOCKER_REGISTRY_PASSWORD‘: Your Nexus password (mark as **Masked**) - ‘DOCKER_REGI‘nexus-docker:5001‘ - ‘DOCKER_IMAGE_NAME‘: ‘java-app‘

Step 3: Verify Runner is Running

```
docker logs -f gitlab-runner
```

Key Configuration Points

DinD Service Setup

```
services:
  - name: docker:24-dind
    alias: docker
    command: ["--tls=false"]  # Reliability without TLS in dev
```

Docker Socket Sharing

```
volumes = [
  "/cache",
  "/var/run/docker.sock:/var/run/docker.sock"  # Access host Docker
]
```

Environment Variables

```
DOCKER_TLS_CERTDIR: ""              # Required for DinD
DOCKER_HOST: tcp://docker:2375      # DinD endpoint
DOCKER_DRIVER: overlay2              # Efficient storage
FF_DOCKER_LAYER_CACHING: "true"     # Reuse build layers
```

How It Works

```
1. GitLab CI/CD triggers docker_build job
   ?
2. Job runs in docker:24-cli container
   ?
3. DinD service (docker:24-dind) provides Docker daemon
   ?
4. Job executes docker build/push commands
   ?
5. Image gets pushed to nexus-docker:5001 registry
```

Docker Build Commands in Your Pipeline

Example in ‘.gitlab-ci.yml‘: “‘yaml script: - echo "$DOCKER_REGISTRY_PASSWORD" | docker login -u "$DOCKER_REGISTRY_USER" --password-stdin $DOCKER_REGISTRY - docker build -t $DOCKER_REGISTRY/$DOCKER_IMAGE_NAME:${CI_COMMIT_SHA:0:8} . - docker push $DOCKER_REGISTRY/$DOCKER_IMAGE_NAME:${CI_COMMIT_SHA:0:8} “‘

Networking

? All components use ‘modernization-network‘ bridge
? Service discovery by container name:
? ‘docker:2375‘ ? DinD daemon
? ‘nexus-docker:5001‘ ? Nexus registry
? ‘gitlab-server‘ ? GitLab instance

Validation

Check runner status: ‘‘‘bash # View runner logs docker logs gitlab-runner

Test Docker connectivity

docker exec gitlab-runner docker ps

List active jobs

docker exec gitlab-runner gitlab-runner --debug list ‘‘‘

Next Steps

1. Register runner with GitLab if not done automatically 2. Set CI/CD variables for Nexus authentication 3. Trigger a test pipeline to verify Docker build works 4. Monitor logs: ‘docker logs -f gitlab-runner‘

Troubleshooting Reference

See files/RUNNER-SETUP.md for detailed troubleshooting: - Runner connection issues - Docker daemon connectivity - Nexus registry authentication - Disk space and memory issues

Security Notes

? **Development**: Current setup is fine for local development ?? **Production**: - Enable TLS for DinD - Don’t share docker.sock (use Kubernetes executor instead) - Use proper registry authentication - Rotate credentials regularly - Add network policies

For production, consider Kubernetes executor with pod security policies.

SOURCE: RUNNER-QUICK-REFERENCE.md

GitLab Runner DinD - Quick Reference Card

? Files Modified/Created

File	Change	Purpose
‘java-pipeline/runner-config.toml‘	?? Modified	DinD runner config with sock
‘java-pipeline/.gitlab-ci.yml‘	?? Modified	Enhanced Docker build job va
‘files/setup-gitlab-runner-dind.ps1‘	? Created	Automated runner setup scrip
‘files/RUNNER-SETUP.md‘	? Created	Complete setup guide & troub
‘files/RUNNER-SETUP-SUMMARY.md‘	? Created	Configuration overview
‘files/gitlab-ci-dind-example.yml‘	? Created	Complete pipeline example

? Quick Start

```
# 1. Run setup script
cd files
.\setup-gitlab-runner-dind.ps1

# 2. Enter registration token when prompted (from GitLab)
# 3. Script will start the runner

# 4. Set CI/CD variables in GitLab UI
#    Settings > CI/CD > Variables:
#    - DOCKER_REGISTRY_USER (your Nexus username)
#    - DOCKER_REGISTRY_PASSWORD (masked)
```

? Key Configuration

DinD Service

```
services:
  - name: docker:24-dind
    alias: docker
    command: ["--tls=false"]
```

Environment Variables

```
DOCKER_TLS_CERTDIR: ""              # Required for DinD
DOCKER_HOST: tcp://docker:2375      # DinD endpoint
DOCKER_DRIVER: overlay2              # Storage driver
```

Runner Settings

```
privileged = true                    # Docker access
volumes = [
  "/var/run/docker.sock:/var/run/docker.sock"  # Socket sharing
]
shm_size = 2147483648               # 2GB shared memory
```

? Verification Steps

```
# 1. Check runner is running
docker ps | grep gitlab-runner

# 2. View logs
docker logs -f gitlab-runner

# 3. Test Docker in runner
docker exec gitlab-runner docker ps

# 4. Trigger test pipeline in GitLab
# Watch the docker_build job in GitLab UI
```

? GitLab CI/CD Variables Required

```
DOCKER_REGISTRY_USER       = <nexus-username>
DOCKER_REGISTRY_PASSWORD   = <nexus-password> [MASKED]
DOCKER_REGISTRY            = nexus-docker:5001
DOCKER_IMAGE_NAME          = java-app
```

? How DinD Works

```
CI Pipeline
   ?
Docker executor (docker:24-cli)
   ?
DinD service (docker:24-dind) provides Docker daemon
   ?
Runner can execute: docker build, docker push, etc.
   ?
Images pushed to Nexus registry (5001)
```

? Common Commands

```
# View runner logs
docker logs gitlab-runner

# Restart runner
docker restart gitlab-runner

# Check Docker daemon in runner
docker exec gitlab-runner docker info

# List active jobs
docker exec gitlab-runner gitlab-runner --debug list

# Rotate runner token (after updating in GitLab)
docker restart gitlab-runner
```

?? Important Notes

1. **TLS Disabled**: Using ‘--tls=false‘ for development reliability 2. **Privileged Mode**: Required for Docker operations in CI 3. **Socket Sharing**: Allows accessing host Docker daemon 4. **Network**: Runner must be on ‘modernization-network‘ 5. **Credentials**: Always mask sensitive variables in GitLab

? Troubleshooting Quick Links

? Runner won’t connect: Check GitLab URL and token in config
? Docker commands fail: Verify ‘docker info‘ works in runner
? Image push fails: Check Nexus credentials and registry URL
? Out of memory: Increase ‘shm_size‘ in config

See ‘files/RUNNER-SETUP.md‘ for detailed troubleshooting.

? Full Documentation

? **Setup Guide**: ‘files/RUNNER-SETUP.md‘
? **Configuration Summary**: ‘files/RUNNER-SETUP-SUMMARY.md‘
? **Pipeline Example**: ‘files/gitlab-ci-dind-example.yml‘
? **Runner Config**: ‘java-pipeline/runner-config.toml‘
? **CI/CD Config**: ‘java-pipeline/.gitlab-ci.yml‘

? Next Steps

1. ? Run ‘setup-gitlab-runner-dind.ps1‘ to create/start runner 2. ? Get registration token from GitLab (Settings > CI/CD > Runners) 3. ? Set Docker registry credentials in GitLab variables 4. ? Push changes to trigger ‘docker_build‘ job 5. ? Verify images appear in Nexus registry