Teleios Free Courses Complete Curriculum

Overview

A comprehensive pathway from complete beginner to Teleios Advanced Program readiness, featuring 10 courses across three progressive pathways. Designed for early-career engineers and career switchers with emphasis on Azure technologies for Microsoft partnership alignment.

Total Duration: 55-75 weeks (14-19 months)

Target Audience: Complete beginners to engineers with 0-2 years experience

Cost: 100% Free

Platform: Self-paced with optional community support

Program Structure

Foundation Pathway (15-21 weeks)

Cloud-agnostic fundamentals preparing students for any DevOps path

1. Developer Toolkit Essentials (4-6 weeks)
2. Infrastructure & Networking Foundations (5-7 weeks)
3. Software Development Fundamentals (6-8 weeks)

DevOps Essentials Pathway (22-30 weeks)

Azure focus begins - building core DevOps skills on Microsoft's cloud platform

1. Containerization with Docker & Azure (5-7 weeks)
2. CI/CD Pipelines with Azure DevOps & GitHub Actions (6-8 weeks)
3. Infrastructure as Code with Terraform & Azure (6-8 weeks)
4. Kubernetes Fundamentals with AKS (7-9 weeks)

Cloud Engineering Pathway (18-24 weeks)

Advanced Azure and production-grade DevOps practices

1. Advanced AKS & GitOps with Argo CD (7-9 weeks)
2. Observability & Monitoring on Azure (6-8 weeks)
3. Production Operations & Career Readiness (5-7 weeks)

FOUNDATION PATHWAY

Course 1: Developer Toolkit Essentials

Duration: 4-6 weeks | **Effort:** 6-8 hours/week

Target: Complete beginners with no technical experience

Goal: Master the essential tools and workflows every developer and DevOps engineer uses daily

Module 1: Command Line Mastery

Lessons:

1. **Introduction to the Terminal**

- Why command line matters

- Terminal vs GUI: when to use each
- Installing and setting up your terminal (Windows Terminal, iTerm2, etc.)
- Basic navigation commands

2. **Working with Files and Directories**

- Creating, moving, copying, deleting files
- File permissions and ownership
- Finding files (find, grep)
- Text manipulation (cat, less, head, tail)

3. **Command Line Productivity**

- Pipes and redirects
- Environment variables
- Command history and shortcuts
- Aliases and shell configuration

4. **Introduction to Shell Scripting**

- What is a shell script?
- Writing your first bash script
- Variables and conditionals
- Loops and functions
- Practical automation examples

Hands-on Labs:

- Lab 1: File system navigation challenge
- Lab 2: Build a backup script
- Lab 3: Create a system monitoring script

Module 2: Version Control with Git

Lessons:

1. **Version Control Fundamentals**

- What is version control and why it matters
- Centralized vs distributed version control
- Git vs other VCS
- Installing and configuring Git

2. **Git Core Concepts**

- Repositories, commits, and the working tree
- The three states: working directory, staging area, repository
- Making your first commit
- Viewing history and differences

3. **Branching and Merging**

- What are branches and why use them?
- Creating and switching branches
- Merging strategies
- Resolving merge conflicts
- Branch naming conventions

4. **Collaboration Workflows**

- Remote repositories
- Pushing and pulling changes
- Fetch vs pull
- Common workflows (feature branch, GitFlow basics)

- .gitignore and best practices

Hands-on Labs:

- Lab 1: Create a local repository and practice commits
- Lab 2: Branch, merge, and resolve conflicts
- Lab 3: Simulate team collaboration workflow

Module 3: GitHub & Collaboration

Lessons:

1. **Introduction to GitHub**

- GitHub vs Git
- Creating your GitHub account
- Profile optimization for developers
- Public vs private repositories

2. **GitHub Workflows**

- Forking repositories
- Cloning and contributing
- Pull requests (PRs)
- Code review basics
- Issues and project management

3. **GitHub Collaboration Features**

- README files and documentation

- GitHub Pages for portfolio sites
- GitHub Gists
- Exploring open source projects

4. **Building Your Developer Profile**

- Contribution graph and activity
- Pinned repositories
- Writing effective commit messages
- Creating a portfolio-ready GitHub profile

Hands-on Labs:

- Lab 1: Create and customize your GitHub profile
- Lab 2: Fork an open-source project and make a contribution
- Lab 3: Collaborate on a team project with PRs and reviews

Module 4: Development Environment Setup

Lessons:

1. **Code Editors and IDEs**

- VS Code setup and configuration
- Essential extensions
- Keyboard shortcuts for productivity
- Integrated terminal usage

2. **Package Managers and Dependencies**

- What are package managers?
- npm/yarn for JavaScript
- pip for Python
- Dependency management basics

3**. Working with Documentation**

- Reading technical documentation effectively
- Stack Overflow and community resources
- Documentation tools (Markdown, README best practices)

4**. Developer Productivity Tools**

- Terminal multiplexers (tmux basics)
- Dotfiles and configuration management
- Command-line tools worth knowing
- Building your toolkit

Hands-on Labs:

- Lab 1: Set up a complete VS Code development environment
- Lab 2: Create a personal dotfiles repository
- Lab 3: Document a project with professional README

Course 1 Capstone Project: Personal Developer Portfolio

Requirements:

- Create a GitHub profile with professional README
- Build 3-5 small projects showcasing different skills

- Document each project thoroughly
- Use proper Git workflow (branches, commits, PRs)
- Deploy at least one project to GitHub Pages

Deliverable: A portfolio website that showcases your GitHub projects

Prerequisites: None - designed for complete beginners

Completion Skills:

- Command line proficiency
- Git and GitHub mastery
- Professional developer workflow
- Documentation skills
- Ready to learn system administration and programming

Course 2: Infrastructure & Networking Foundations

Duration: 5-7 weeks | **Effort:** 7-9 hours/week

Target: Students who completed Course 1 or have basic command line/Git knowledge

Goal: Understand how computers, networks, and servers work - the foundation for any infrastructure role

Module 1: Linux Operating System

Lessons:

1. **Introduction to Linux**

- Linux history and philosophy

- Distributions: Ubuntu, Debian, CentOS, RHEL
- Desktop vs Server editions
- Installing Linux (dual boot, VM, WSL)

2. **Linux File System**

- File system hierarchy (/, /home, /etc, /var, etc.)
- File types and permissions (rwx)
- chmod, chown, and access control
- Links (hard vs symbolic)

3. **User and Process Management**

- Users and groups
- sudo and privilege escalation
- Process lifecycle
- ps, top, htop, kill commands
- Background and foreground processes

4. **Package Management**

- APT (Ubuntu/Debian)
- YUM/DNF (RHEL/CentOS)
- Installing, updating, removing packages
- Repository management
- Building software from source

Hands-on Labs:

- Lab 1: Install Ubuntu Server in a VM
- Lab 2: User management and permissions scenario
- Lab 3: System monitoring and process management

Module 2: Networking Fundamentals

Lessons:

1. **Networking Basics**

- What is a network?
- IP addresses (IPv4 and IPv6)
- MAC addresses
- Subnets and CIDR notation
- Public vs private IP addresses

2. **Network Protocols**

- OSI Model overview
- TCP vs UDP
- Common protocols (HTTP/HTTPS, SSH, FTP, DNS)
- Port numbers and services
- Three-way handshake

3. **DNS and Domain Names**

- How DNS works
- DNS record types (A, AAAA, CNAME, MX, TXT)
- DNS resolution process
- Using dig and nslookup
- DNS propagation

4. **Network Troubleshooting**

- ping and connectivity testing
- traceroute/tracert
- netstat and ss
- nmap basics
- Reading network logs

Hands-on Labs:

- Lab 1: Network calculator and subnetting practice
- Lab 2: Set up a local DNS server
- Lab 3: Network troubleshooting scenarios

Module 3: Web Servers and HTTP

Lessons:

1. **How the Web Works**

- Client-server architecture
- HTTP request/response cycle
- Status codes (200, 404, 500, etc.)
- Headers and cookies
- HTTPS and SSL/TLS basics

2. **Web Server Software**

- Apache vs Nginx
- Installing and configuring Nginx
- Virtual hosts and server blocks
- Static file serving

- Log files and analysis

3. **Reverse Proxies and Load Balancing**

- What is a reverse proxy?
- Nginx as a reverse proxy
- Load balancing concepts
- Health checks
- SSL termination

4. **Web Server Security**

- Firewall basics (ufw, iptables)
- SSH hardening
- Fail2ban
- Security headers
- Rate limiting

Hands-on Labs:

- Lab 1: Set up Nginx web server
- Lab 2: Configure reverse proxy for multiple applications
- Lab 3: Implement basic security measures

Module 4: System Administration Basics

Lessons:

1. **Server Management**

- Remote access with SSH
- SSH keys vs passwords
- SSH config files
- SCP and SFTP for file transfer
- Systemd and service management

2. **Logging and Monitoring**

- Linux log files (/var/log)
- journalctl and systemd logs
- Syslog
- Basic monitoring (disk space, memory, CPU)
- Setting up basic alerts

3. **Backup and Recovery**

- Backup strategies (full, incremental, differential)
- rsync for backups
- Tar and compression
- Disaster recovery planning
- Testing backups

4. **Automation and Cron Jobs**

- Cron syntax and scheduling
- Anacron for non-24/7 systems
- Automating system maintenance
- Log rotation
- System updates automation

Hands-on Labs:

- Lab 1: Set up automated backups with rsync
- Lab 2: Create system monitoring script with cron
- Lab 3: Implement log rotation and cleanup

Course 2 Capstone Project: Self-Hosted Infrastructure

Requirements:

- Set up 2-3 Linux VMs (can use VirtualBox/VMware locally)
- Configure networking between VMs
- Deploy multiple web applications
- Set up Nginx as reverse proxy/load balancer
- Implement monitoring and logging
- Configure automated backups
- Document your infrastructure with network diagrams
- Create runbooks for common tasks

Prerequisites: Course 1 or equivalent command line/Git experience

Completion Skills:

- Linux system administration
- Networking concepts and troubleshooting
- Web server configuration
- Basic security practices
- Ready to learn application development

Course 3: Software Development Fundamentals

Duration: 6-8 weeks | **Effort:** 8-10 hours/week

Target: Students with command line, Git, and basic Linux knowledge

Goal: Understand how modern applications are built - essential knowledge for deploying and managing them

Module 1: Programming Fundamentals with Python

Lessons:

1. **Python Basics**

- Why Python for DevOps?
- Installing Python and pip
- Variables, data types, and operators
- Input/output
- Comments and code style (PEP 8)

2. **Control Flow and Functions**

- Conditionals (if/elif/else)
- Loops (for, while)
- Functions and parameters
- Return values
- Scope and namespaces

3. **Data Structures**

- Lists and list comprehensions
- Dictionaries and sets
- Tuples
- Working with JSON

- File I/O

4. **Python for Automation**

- Working with files and directories (os, pathlib)
- Running system commands (subprocess)
- Regular expressions basics
- Error handling and exceptions
- Building CLI tools with argparse

Hands-on Labs:

- Lab 1: System administration scripts (user management, file organization)
- Lab 2: Log parser and analyzer
- Lab 3: Build a CLI tool for common tasks

Module 2: APIs and Web Development Basics

Lessons:

1. **Understanding APIs**

- What is an API?
- REST principles
- HTTP methods (GET, POST, PUT, DELETE)
- Request/response structure
- JSON and data formats
- API authentication (API keys, tokens)

2. **Working with APIs in Python**

- Requests library
- Making API calls
- Parsing JSON responses
- Error handling
- Rate limiting
- Building a simple API client

3. **Web Application Basics**

- Frontend vs Backend
- HTML/CSS overview (what you need to know)
- JavaScript basics (for DevOps context)
- How web frameworks work
- MVC pattern

4. **Building a Simple Web App**

- Flask framework introduction
- Routes and views
- Templates basics
- Forms and POST requests
- Serving static files
- Application structure

Hands-on Labs:

- Lab 1: Build an API client for a public API (GitHub, weather, etc.)
- Lab 2: Create a simple Flask application
- Lab 3: Build a dashboard that consumes an API

Module 3: Databases and Data Persistence

Lessons:

1. **Database Fundamentals**

- Why databases?
- Relational vs NoSQL
- ACID properties
- Database design basics
- ERD diagrams

2. **SQL and Relational Databases**

- Installing PostgreSQL/MySQL
- SQL basics (SELECT, INSERT, UPDATE, DELETE)
- Filtering and sorting
- Joins (INNER, LEFT, RIGHT)
- Indexes and performance
- Transactions

3. **Working with Databases in Python**

- Database drivers (psycopg2, mysql-connector)
- Connection management
- Parameterized queries (SQL injection prevention)
- ORM basics (SQLAlchemy introduction)
- Database migrations concept

4. **NoSQL Basics**

- Document databases (MongoDB concepts)

- Key-value stores (Redis concepts)
- When to use NoSQL
- CAP theorem introduction

Hands-on Labs:

- Lab 1: Design and implement a database schema
- Lab 2: Build Python scripts with database operations
- Lab 3: Add database to your Flask application

Module 4: Application Architecture and Best Practices

Lessons:

1. **Application Configuration**

- Environment variables
- Configuration files (YAML, JSON, INI)
- Secrets management basics
- 12-factor app principles
- Configuration per environment

2. **Application Structure and Organization**

- Project layout best practices
- Separation of concerns
- Modules and packages
- Dependency management
- Virtual environments (venv, virtualenv)

3. **Testing Basics**

- Why testing matters
- Unit tests vs integration tests
- pytest introduction
- Writing testable code
- Test coverage

4. **Application Deployment Basics**

- Application dependencies
- Requirements.txt and lock files
- WSGI servers (gunicorn, uWSGI)
- Process management (systemd)
- Application logs
- Health checks and readiness

Hands-on Labs:

- Lab 1: Refactor an application with proper structure
- Lab 2: Write tests for your application
- Lab 3: Deploy application with proper process management

Course 3 Capstone Project: Full-Stack Application with Infrastructure

Requirements:

Build a complete application demonstrating all skills:

- **Backend:** Python/Flask API with PostgreSQL database
- **Features:** User authentication, CRUD operations, API endpoints
- **Infrastructure:** Self-hosted on Linux server
- **Configuration:** Environment-based configuration
- **Testing:** Unit and integration tests
- **Deployment:** Automated deployment script
- **Monitoring:** Health checks, logging, basic monitoring
- **Documentation:** API docs, setup instructions, architecture diagram
- **Bonus:** Add a simple frontend or CLI client

Example Projects:

- Task management API with team collaboration
- School management system
- E-commerce product catalog
- Blog platform with comments

Prerequisites: Courses 1 & 2 or equivalent

Completion Skills:

- Programming proficiency (Python)
- API development and consumption
- Database design and management
- Application deployment
- Testing and quality practices
- **READY FOR DEVOPS ESSENTIALS PATHWAY**

DEVOPS ESSENTIALS PATHWAY

Course 4: Containerization with Docker & Azure

Duration: 5-7 weeks | **Effort:** 8-10 hours/week

Target: Students who completed Foundation Pathway or have equivalent programming and Linux skills

Goal: Master containerization concepts and practices, from local development to production deployment on Azure

Module 1: Container Fundamentals

Lessons:

1. **Introduction to Containers**

- What are containers and why they matter
- Containers vs Virtual Machines
- Docker architecture (daemon, client, registry)
- Installing Docker Desktop and Docker Engine
- Docker basics: images, containers, volumes, networks

2. **Working with Docker Images**

- Pulling images from Docker Hub
- Running your first container
- Container lifecycle (create, start, stop, remove)
- Inspecting containers and logs
- Interactive vs detached mode
- Port mapping and networking basics

3. **Building Custom Images**

- Dockerfile syntax and structure
- FROM, RUN, COPY, CMD, ENTRYPOINT

- Building images with docker build
- Image layers and caching
- Tagging and versioning images
- .dockerignore files

4**. Docker Best Practices**

- Multi-stage builds
- Minimizing image size
- Using official base images
- Security considerations
- Non-root users in containers
- Health checks

Hands-on Labs:

- Lab 1: Run and manage existing containers (nginx, postgres, redis)
- Lab 2: Create a Dockerfile for a Python Flask application
- Lab 3: Optimize an image using multi-stage builds

Module 2: Docker Compose and Multi-Container Applications

Lessons:

1**. Docker Compose Fundamentals**

- What is Docker Compose?
- YAML syntax basics
- docker-compose.yml structure
- Services, networks, and volumes

- docker-compose commands

2. **Building Multi-Container Applications**

- Frontend + Backend + Database architecture
- Service dependencies (depends_on)
- Environment variables and .env files
- Named volumes for data persistence
- Custom networks for service isolation

3. **Development Workflows with Compose**

- Hot reload and volume mounting
- Override files for different environments
- Scaling services
- Viewing logs across services
- Debugging containerized applications

4. **Docker Networking Deep Dive**

- Bridge networks
- Host and none networks
- Custom networks
- Service discovery and DNS
- Container-to-container communication
- Exposing services externally

Hands-on Labs:

- Lab 1: Create a docker-compose.yml for a 3-tier application
- Lab 2: Set up development environment with hot reload
- Lab 3: Implement service-to-service communication

Module 3: Azure Container Registry (ACR)

Lessons:

1. **Introduction to Azure and Container Registries**

- Creating your Azure free account
- Azure Portal navigation
- What is a container registry?
- Docker Hub vs private registries
- ACR tiers and features

2. **Setting Up Azure Container Registry**

- Creating an ACR instance
- Azure CLI installation and configuration
- Authentication methods (admin, service principal, managed identity)
- Repository structure and naming
- ACR tasks introduction

3. **Pushing and Managing Images in ACR**

- Tagging images for ACR
- docker login to ACR
- Pushing images to ACR
- Listing and managing repositories
- Image scanning and security
- Retention policies and cleanup

4. **CI/CD Integration with ACR**

- ACR webhooks
- ACR tasks for automated builds
- Building images in the cloud
- Multi-architecture images
- Geo-replication for global access

Hands-on Labs:

- Lab 1: Set up ACR and push your first image
- Lab 2: Implement automated builds with ACR Tasks
- Lab 3: Set up webhooks for deployment notifications

Module 4: Container Deployment on Azure

Lessons:

1. **Azure Container Instances (ACI)**

- What is ACI and when to use it
- Creating container instances via Portal and CLI
- Resource allocation (CPU, memory)
- Public vs private IP addresses
- Pulling from ACR

2. **Deploying Multi-Container Applications**

- Container groups in ACI
- YAML deployment definitions
- Shared volumes between containers

- Environment variables and secrets
- Restart policies

3**. Azure App Service for Containers**

- App Service overview
- Deploying containers to App Service
- Continuous deployment from ACR
- Custom domain and SSL
- Scaling and performance
- Logs and monitoring with Application Insights

4**. Production Considerations**

- Container security best practices
- Secret management (Azure Key Vault integration)
- Logging and monitoring containers
- Cost optimization
- When to use ACI vs App Service vs AKS
- Health checks and readiness probes

Hands-on Labs:

- Lab 1: Deploy a container to ACI
- Lab 2: Deploy multi-container application to ACI
- Lab 3: Deploy containerized app to App Service with CI/CD

Course 4 Capstone Project: Microservices E-Commerce Platform

Application Components:

- **Frontend Service:** React/Vue.js application (provided template)
- **Product Service:** Python/Flask API managing product catalog
- **Order Service:** Node.js/Express API managing orders
- **Database:** PostgreSQL for products, MongoDB for orders
- **Cache:** Redis for session management

Requirements:

1. Create Dockerfiles for all services with multi-stage builds
2. Create docker-compose.yml for local development
3. Set up Azure Container Registry
4. Push all images to ACR
5. Deploy to Azure Container Instances with proper networking
6. Implement health checks for all services
7. Configure environment-based configuration (dev, prod)
8. Set up logging and basic monitoring
9. Document architecture with diagrams
10. Create README with setup and deployment instructions

Deliverables:

- GitHub repository with all code and Dockerfiles
- Architecture diagram showing container communication
- Deployment scripts for Azure
- Running application accessible via public URL
- Documentation for running locally and deploying

Prerequisites: Course 3 or equivalent programming and Linux skills

Completion Skills:

- Docker image creation and management
- Multi-container application orchestration

- Azure Container Registry management
- Container deployment on Azure
- Security and best practices
- Ready for CI/CD automation

Course 5: CI/CD Pipelines with Azure DevOps & GitHub Actions

Duration: 6-8 weeks | **Effort:** 9-11 hours/week

Target: Students who completed Course 4 or have Docker/containerization experience

Goal: Build automated CI/CD pipelines that test, build, and deploy applications to Azure with confidence

Module 1: CI/CD Principles and Strategy

Lessons:

1. **Continuous Integration Fundamentals**

- What is Continuous Integration?
- Benefits and challenges of CI
- CI best practices
- Trunk-based development
- Feature flags and toggles
- The deployment pipeline concept

2. **Continuous Delivery vs Continuous Deployment**

- Defining Continuous Delivery
- Continuous Deployment differences
- Deployment strategies (blue-green, canary, rolling)

- Release management concepts
- Rollback strategies

3**. Git Workflows for CI/CD**

- GitFlow workflow
- GitHub Flow
- Trunk-based development
- Branch protection rules
- Pull request workflows
- Semantic versioning

4**. Testing Strategy in Pipelines**

- Test pyramid concept
- Unit tests, integration tests, e2e tests
- Test coverage metrics
- Performance testing
- Security testing (SAST, DAST)
- When to run different test types

Hands-on Labs:

- Lab 1: Set up branch protection and PR workflows
- Lab 2: Implement semantic versioning automation
- Lab 3: Design a deployment strategy for a web application

Module 2: GitHub Actions Deep Dive

Lessons:

1. **GitHub Actions Fundamentals**

- What are GitHub Actions?
- Workflows, jobs, and steps
- Events and triggers
- GitHub-hosted vs self-hosted runners
- Actions marketplace
- Workflow syntax (YAML)

2. **Building CI Pipelines with GitHub Actions**

- Checkout and setup actions
- Building and testing code
- Running tests in parallel
- Code quality checks (linting, formatting)
- Caching dependencies
- Artifact management

3. **Advanced GitHub Actions**

- Matrix builds (multiple versions, OS)
- Reusable workflows
- Composite actions
- Custom actions (JavaScript, Docker)
- Workflow templates
- Status badges

4. **Secrets and Environment Management**

- GitHub Secrets
- Environment secrets
- Environment protection rules
- Using Azure credentials in workflows

- OpenID Connect (OIDC) with Azure
- Secret scanning

Hands-on Labs:

- Lab 1: Build a CI workflow with testing and linting
- Lab 2: Create a matrix build for multiple platforms
- Lab 3: Build a reusable workflow for common tasks

Module 3: Azure Pipelines Mastery

Lessons:

1. **Azure DevOps Overview**

- Azure DevOps services overview
- Creating an Azure DevOps organization
- Projects and repositories
- Azure Repos vs GitHub integration
- Boards, Test Plans, Artifacts

2. **Azure Pipelines Fundamentals**

- Classic vs YAML pipelines
- Pipeline structure and syntax
- Agents and agent pools
- Microsoft-hosted vs self-hosted agents
- Variables and variable groups
- Pipeline triggers

3**. Building Multi-Stage Pipelines**

- Stages, jobs, and tasks
- Dependencies between stages
- Conditions and expressions
- Templates and extends
- Pipeline artifacts
- Container jobs

4**. Azure Pipeline Integrations**

- Service connections to Azure
- Integration with Azure Container Registry
- Azure Key Vault integration
- Approval gates and checks
- Pipeline decorators
- Release management

Hands-on Labs:

- Lab 1: Create a basic Azure Pipeline for a containerized app
- Lab 2: Build a multi-stage pipeline with test and deploy stages
- Lab 3: Implement manual approval gates

Module 4: Deployment Automation and Best Practices