**Here’s a structured roadmap for you 👇**

**🛠 DevOps Roadmap for a System Administrator**

1. Strengthen Your Foundations (you already have partial skills)

✅ You already know: OS, servers, networking, troubleshooting.  
🔹 Add/Improve on:

* Linux basics → shell scripting (bash), package management, systemd, monitoring.
* Windows basics → PowerShell scripting, IIS, Active Directory (you probably know this).
* Networking → DNS, Load Balancers, Reverse Proxies (Nginx/HAProxy).

**2. Learn Version Control (Git + GitHub/Azure Repos)**

* Understand git clone, commit, push, branch, merge, rebase.
* Practice with small projects (you’re already doing this ✅).
* Learn Git branching strategies (GitFlow, Trunk-based).

**3. Build CI/CD Skills (Azure DevOps, GitHub Actions, Jenkins)**

* Learn YAML pipelines (you already started 💯).
* Write simple pipelines: build → test → publish artifacts.
* Learn multi-stage pipelines (Build → Test → Deploy).
* Get hands-on with Azure DevOps Pipelines (since you’re already inside Azure).

**4. Configuration Management**

* Learn Infrastructure as Code (IaC) tools:
  + Ansible → Automate server configurations.
  + Terraform → Provision cloud infra (Azure/AWS).
  + ARM/Bicep templates → Azure-specific infra automation.

**5. Containers & Orchestration**

* Learn Docker: build images, run containers.
* Learn Kubernetes basics: pods, services, deployments.
* Use AKS (Azure Kubernetes Service) for practice.

**6. Cloud Skills (Azure recommended for you)**

* Learn Azure basics:
  + Azure VMs, Storage, Networking.
  + Azure DevOps, Azure CLI, Resource Groups.
* Learn Azure PaaS:
  + App Service, Functions, Azure SQL.
* Practice deploying apps from your pipeline → Azure App Service.

**7. Monitoring & Logging**

* Learn Prometheus + Grafana or Azure Monitor + Log Analytics.
* Understand how to collect logs, create dashboards, set alerts.

**8. Security & Best Practices**

* Secrets management → Azure Key Vault.
* Secure CI/CD → no passwords in code.
* Role-based access (least privilege).

**🎯 How You Can Progress Step by Step**

1. **Short-term (1–3 months):**
   * Master Git + Azure DevOps pipelines (CI/CD).
   * Learn basic scripting (bash + PowerShell).
   * Deploy small apps to Azure App Service with pipelines.
2. **Medium-term (3–6 months):**
   * Learn Terraform/Ansible (infra automation).
   * Build pipelines that provision infra + deploy apps.
   * Learn Docker basics.
3. **Long-term (6–12 months):**
   * Learn Kubernetes (AKS).
   * Integrate monitoring + alerting into pipelines.
   * Work on real-world DevOps projects (infra as code + CI/CD + containers).

**📚 Recommended Resources**

* Git → [learngitbranching](https://learngitbranching.js.org/) (interactive).
* Azure DevOps → Microsoft Learn modules (free).
* Terraform → HashiCorp Learn.
* Docker → Play with Docker labs.
* Kubernetes → Kubernetes by Example.

**Devops**

DevOps is a combination of "Development" and "Operations"—a culture, set of practices, and tools that bring together software development and IT operations teams to:

* Shorten the software development lifecycle, and
* Deliver high-quality software continuously (known as CI/CD – Continuous Integration/Continuous Delivery).

🔑 **Key Concepts in DevOps**

| **Concept** | **Description** |
| --- | --- |
| Collaboration | Developers and operations teams work together instead of in silos. |
| Automation | Repetitive tasks (e.g., testing, deployment, monitoring) are automated. |
| CI/CD | Continuous Integration & Continuous Delivery – ensures faster and reliable releases. |
| Monitoring | Systems are continuously monitored to detect and respond to issues quickly. |
| Infrastructure as Code (IaC) | Infrastructure is managed using code (e.g., Terraform, Ansible). |

🧰 Common DevOps Tools

| **Function** | **Tools** |
| --- | --- |
| Version Control | Git, GitHub, GitLab |
| CI/CD | Jenkins, GitHub Actions, Azure DevOps, GitLab CI |
| Configuration Management | Ansible, Puppet, Chef |
| Containerization | Docker |
| Orchestration | Kubernetes |
| Monitoring | Prometheus, Grafana, ELK Stack |
| Cloud Platforms | AWS, Azure, GCP |

**📈 Benefits of DevOps**

* Faster delivery of features and fixes
* Improved collaboration and efficiency
* Better product quality and reliability
* Reduced deployment failures and rollback time
* Continuous feedback and monitoring

**🌍 Real-Life Example**

Instead of a developer writing code and "throwing it over the wall" to the operations team to deploy, DevOps ensures:

* Code is automatically tested and deployed via pipelines.
* Systems are monitored in real-time.
* Any issues are quickly fixed and deployed again with minimal downtime.

**🔧** **What is Git?**

**Git is a version control system — it helps you:**

* Track changes in your code
* Go back to earlier versions if something breaks
* Work with others on the same project without conflicts

Think of it like a save system for your code — you can create “checkpoints” (called *commits*) and switch between them.

**🔸 Example:**

* You’re writing a program.
* You make a change and save it using Git.
* If the code breaks later, you can go back to a working version.

**🌐 What is GitHub?**

GitHub is a website where you can:

* Store your Git projects online
* Share your code with others
* Work together with a team
* Showcase your projects to the world

GitHub uses Git behind the scenes, but adds online tools like:

* Pull Requests (for code review)
* Issues (for bug tracking)
* Actions (for automation and CI/CD)

**🔸 Example:**

* You build a website using Git.
* You upload (push) it to GitHub.
* Your friend clones it, adds features, and sends you a pull request.

Here's a **Git Command Handbook for Beginners**—a concise and practical guide to help you understand and use Git effectively.

**🧰 Git Command Handbook for Beginners**

**🔹 1. Setup and Configuration**

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"

git config --list # Check your configuration

**🔹 2. Starting a Git Repository**

git init # Initialize a local repo

git clone <repo\_url> # Clone from a remote repo (GitHub, GitLab, etc.)

**🔹 3. Basic Workflow**

git status # Check the current status of files

git add <file> # Stage a specific file

git add . # Stage all changes

git commit -m "Message" # Commit staged files with a message

**🔹 4. Checking History**

git log # Show full commit history

git log --oneline # Show compact history

git show <commit\_id> # Show details of a specific commit

**🔹 5. Branching**

git branch # List all branches

git branch <branch\_name> # Create a new branch

git checkout <branch\_name> # Switch to a branch

git checkout -b <branch\_name> # Create and switch in one step

**🔹 6. Merging**

git merge <branch\_name> # Merge a branch into current

**🔹 7. Remote Repositories**

git remote -v # View remote connections

git remote add origin <url> # Add a remote repo

git push -u origin main # Push initial code to main branch

git fetch # Fetch changes from remote

git pull # Pull updates from remote

**🔹 8. Undoing Changes**

git checkout -- <file> # Discard local changes

git reset <file> # Unstage a file

git reset --hard # Reset all changes (Caution!)

**🔹 9. Stashing Changes**

git stash # Save changes temporarily

git stash list # Show stashed changes

git stash apply # Re-apply the latest stash

**🔹 10. Tagging**

git tag # List tags

git tag v1.0 # Create a lightweight tag

git tag -a v1.0 -m "Version" # Create an annotated tag

git push origin v1.0 # Push a tag to remote

**📝 Useful Tips**

* .gitignore: Ignore files/folders in Git.
* git diff: See what’s changed before committing.
* git blame <file>: See who last changed each line.
* git reflog: View recent history including orphaned commits.

**Git Command Cheat Sheet for Beginners**

**1. Configuration**

|  |  |
| --- | --- |
| Command | Explanation |
| git config --global user.name "Your Name" | Set your Git username globally. |
| git config --global user.email "you@example.com" | Set your Git email globally. |
| git config --list | View all config settings. |

**2. Repository Setup**

|  |  |
| --- | --- |
| Command | Explanation |
| git init | Create a new Git repository. |
| git clone <repo\_url> | Copy a remote repo locally. |

**3. Staging & Commit**

|  |  |
| --- | --- |
| Command | Explanation |
| git status | Check file changes (staged/untracked). |
| git add <file> | Stage a specific file. |
| git add . | Stage all modified files. |
| git commit -m "message" | Commit staged changes with a message. |

**4. Viewing Changes**

|  |  |
| --- | --- |
| Command | Explanation |
| git log | Show full commit history. |
| git log --oneline | Show brief commit history. |
| git show <commit> | Show details of a specific commit. |
| git diff | See unstaged file changes. |
| git diff --staged | See staged vs. last commit differences. |

**5. Branching**

|  |  |
| --- | --- |
| Command | Explanation |
| git branch | List all branches. |
| git branch <branch\_name> | Create a new branch. |
| git checkout <branch\_name> | Switch to a branch. |
| git checkout -b <branch\_name> | Create and switch to a new branch. |
| git branch -d <branch\_name> | Delete a branch (safe). |
| git branch -D <branch\_name> | Force delete a branch. |

**6. Merge & Rebase**

|  |  |
| --- | --- |
| Command | Explanation |
| git merge <branch> | Merge branch into current. |
| git rebase <branch> | Reapply commits on a new base. |

**7. Remote Repositories**

|  |  |
| --- | --- |
| Command | Explanation |
| git remote -v | View linked remote repos. |
| git remote add origin <url> | Link to a remote repo. |
| git push -u origin main | Push to remote and set upstream. |
| git push | Push local commits to remote. |
| git pull | Fetch and merge from remote. |
| git fetch | Fetch updates without merging. |

**8. Undoing Changes**

|  |  |
| --- | --- |
| Command | Explanation |
| git checkout -- <file> | Discard local file changes. |
| git reset <file> | Unstage a file. |
| git reset --hard | Reset all to last commit (destructive). |

**9. Stashing**

|  |  |
| --- | --- |
| Command | Explanation |
| git stash | Temporarily save changes. |
| git stash list | Show all stashed changes. |
| git stash apply | Reapply last stash (keep stash). |
| git stash pop | Reapply and delete last stash. |

**10. Tagging**

|  |  |
| --- | --- |
| Command | Explanation |
| git tag | List all tags. |
| git tag v1.0 | Create a lightweight tag. |
| git tag -a v1.0 -m "Message" | Create annotated tag with message. |
| git push origin v1.0 | Push specific tag to remote. |
| git push --tags | Push all tags to remote. |

**11. Clean Up**

|  |  |
| --- | --- |
| Command | Explanation |
| git clean -f | Remove untracked files. |
| git clean -fd | Remove untracked files and directories. |

**12. Inspection**

|  |  |
| --- | --- |
| Command | Explanation |
| git blame <file> | Show who last modified each line. |
| git reflog | Show history including detached commits. |

**13. Ignore Files**

|  |  |
| --- | --- |
| File | Explanation |
| .gitignore | File to specify what to ignore in Git. (e.g., \*.log, node\_modules/) |

**✅ What is CI/CD?**

**CI/CD** stands for:

* **CI** = **Continuous Integration**
* **CD** = **Continuous Delivery** *(or)* **Continuous Deployment**

It is a **modern DevOps practice** used to automate and streamline the software **build → test → release** process.

**🔹 CI – Continuous Integration**

**Purpose:**  
Automatically build and test code whenever developers push changes to a shared repository.

**🔧 Key Activities:**

* Code is pushed (Git commit)
* Automated build is triggered
* Automated unit/integration tests are run
* Developers are notified of build/test results

**✅ Benefits:**

* Early bug detection
* Faster feedback loop
* Improved code quality
* Easier integration across teams

**🔹 CD – Continuous Delivery / Deployment**

There are **two types of CD**:

**1. Continuous Delivery**

* Automatically prepares code for **release to production**
* Final **manual approval** before production deploy

**2. Continuous Deployment**

* Goes one step further: **automatically deploys to production**
* **No manual approval** needed

**🚀 Activities:**

* Create release package or artifact
* Deploy to Dev → QA → Staging → Production
* Run post-deployment checks

**🔄 CI/CD Pipeline Flow Diagram:**

Developer Pushes Code

|

v

[ Continuous Integration ]

|

Code Build & Unit Test

|

v

[ Continuous Delivery ]

|

Deploy to Staging

|

Manual Approval

|

v

[ Continuous Deployment ]

Deploy to Production

**🧪 Real-world Tools for CI/CD:**

| **Stage** | **Tools** |
| --- | --- |
| Source Code | Git, GitHub, GitLab, Azure Repos |
| CI/CD | **Azure DevOps**, GitHub Actions, Jenkins, GitLab CI, CircleCI |
| Build/Test | Maven, Gradle, npm, dotnet, JUnit, Pytest |
| Deploy | Azure Pipelines, AWS CodeDeploy, Helm, Terraform, Kubernetes |

**💡 Example with Azure DevOps:**

* Dev pushes code to main branch
* Azure Pipeline triggers (CI)
* App is built and tested
* Artifact is created and pushed (CD)
* Optionally auto-deploys to Azure Web App or Kubernetes

**✅ Summary**

| **Term** | **Description** |
| --- | --- |
| **CI** | Automate **code integration, build, and test** |
| **CD (Delivery)** | Automate **packaging and release preparation** |
| **CD (Deployment)** | Automate **deployment to production** |

**Azure DevOps YAML Pipeline Structure (CI/CD)**

| **Section** | **Purpose** |
| --- | --- |
| trigger | Define branch-based pipeline triggers |
| variables | Declare global variables used across pipeline |
| pool | Define agent VM image |
| stages | Top-level logical stages in your CI/CD pipeline |
| jobs | Group of steps run on a single agent inside a stage |
| steps | Individual actions: script or task |
| task | Predefined Azure action (e.g., DotNetCoreCLI, AzureWebApp) |
| script | Custom shell commands |

**Below Script to understand**

**🚀 Simple Azure Pipeline YAML Example**

This script:

* Triggers on push to the main branch
* Uses a **Linux agent**
* Installs **Node.js**
* Runs **build** and **test** scripts from package.json

# azure-pipelines.yml

trigger:

branches:

include:

- main # Trigger when code is pushed to main branch

pool:

vmImage: 'ubuntu-latest' # Use Microsoft-hosted Linux agent

steps:

- task: NodeTool@0 # Install Node.js

inputs:

versionSpec: '18.x'

displayName: 'Install Node.js'

- script: |

npm install # Install dependencies

displayName: 'Install NPM Packages'

- script: |

npm run build # Run custom build script

displayName: 'Build Application'

- script: |

npm test # Run test cases

displayName: 'Run Tests'

**🪜 Step-by-Step Guide to create basic pipeline**

**Step 1: Prepare Your Code Repository**

**Make sure your repo has:**

* **A package.json file in the root directory.**
* **Sample content (minimal example):**

**{**

**"name": "simple-node-app",**

**"version": "1.0.0",**

**"scripts": {**

**"build": "echo Build completed",**

**"test": "echo All tests passed"**

**}**

**}**

**Step 2: Sign In to Azure DevOps**

1. **Go to 👉 https://dev.azure.com**
2. **Open your organization and project**

**Step 3: Create a New Pipeline**

1. **In your Azure DevOps Project, go to Pipelines > Pipelines**
2. **Click “New Pipeline”**
3. **Choose your source:**
   * **Azure Repos Git (or GitHub, Bitbucket)**
4. **Select the repository with your Node.js app**

**Step 4: Configure the Pipeline**

1. **Choose YAML**
2. **When prompted, Azure may suggest a template — click “Starter pipeline”**
3. **Delete the default content**
4. **Paste the below YAML:**

**trigger:**

**branches:**

**include:**

**- main**

**pool:**

**vmImage: 'ubuntu-latest'**

**steps:**

**- task: NodeTool@0**

**inputs:**

**versionSpec: '18.x'**

**displayName: 'Install Node.js'**

**- script: |**

**npm install**

**displayName: 'Install NPM Packages'**

**- script: |**

**npm run build**

**displayName: 'Build Application'**

**- script: |**

**npm test**

**displayName: 'Run Tests'**

**Step 5: Save and Run the Pipeline**

1. **Click “Save and run”**
2. **Commit message can be: Add CI pipeline**
3. **Click Save and run again**

**🔧 Azure will:**

* **Save the YAML file as azure-pipelines.yml in your repo**
* **Queue and run your pipeline immediately**

**Step 6: View the Results**

* **Watch each step execute in real-time**
* **A green check ✅ means it passed**
* **A red cross ❌ means it failed (click to see the logs)**