Module 2: Source Code Management (Repos)

1. Creating Organizations & Projects in Azure DevOps

1.1 Azure DevOps Organization

An **Organization** is the top-level container in Azure DevOps.
It represents the company, department, or business unit using Azure DevOps.

Key Capabilities of an Organization

- Houses one or more **projects**
- Uses Azure Active Directory (Entra ID) or Microsoft Account for authentication
- Central location for:
    - User management
    - Billing
    - Extension configuration
    - Security policies
    - Cross-project governance

Steps: Create an Azure DevOps Organization

1. Open: **https://dev.azure.com/**
2. Sign in with Microsoft/Entra ID account.
3. Click **New Organization**.
4. Provide:
    - Organization name
    - Region (important for performance + compliance)
5. Click **Continue**.

Best Practice:
Choose a region closest to the majority of developers.

1.2 Azure DevOps Projects

A **Project** is a logical container for everything related to a software product or team.

Each project contains:

- Boards (Agile Work Items)

- Repos (Git / TFVC)
- Pipelines
- Test Plans
- Artifacts

Projects isolate:

- Permissions
- Repositories
- Pipelines
- Work tracking
- Team structures

Steps: Create a New Project

1. Inside your Organization → Click **New Project**
2. Enter:
    - Project Name
    - Description
    - Visibility (Private or Public)
    - Version Control: Git or TFVC
    - Work item process: Agile/Scrum/CMMI/Basic
3. Click **Create**.

Best Practice:
Use **one project per product/team**, not per environment.

---

2. Importing Repositories (Git / TFVC)

2.1 Importing a Git Repository

You can import:

- GitHub repo
- Bitbucket
- GitLab
- Any external Git source

Steps: Import Git Repo

1. Go to **Repos → Files**
2. Click **Import a Repository**
3. Provide:
    - Clone URL (HTTPS)
    - Authentication if required (PAT/username-password)
4. Click **Import**.

Important Notes

- Branches and tags are imported automatically.
- Private repos need a **Personal Access Token (PAT)**.

2.2 Migrating TFVC to Git (Optional Enterprise Use Case)

While TFVC is still supported for legacy apps, enterprises are moving to Git.

TFVC → Git Migration Approaches

- Partial migration (selected history)
- Full migration (entire history)
- Using **git-tf** or **Import Tool**

---

3. Branching Strategies (Main/Develop/Feature)

Branching strategy defines **how teams write, review, test, and merge code**.

3.1 GitFlow Model (Classic Enterprise Model)

main → production-ready code

develop → integration branch for upcoming release

feature/xxx → feature branches

hotfix/xxx → urgent fixes

release/xxx → pre-production stabilization

Flow

Feature → Develop → Release → Main

3.2 Trunk-Based Development (Modern DevOps Recommended)

main → single trunk

feature branches → short-lived

PR → main (multiple times per day)

Benefits

- Faster CI/CD
- Avoids long-lived branches
- Suitable for microservices

3.3 Three-Branch Strategy (Simple & Popular)

main   → stable production

develop   → upcoming features integration

feature/* → developer work branches

Use When

Medium teams (10–40 developers) with scheduled releases.

---

4. Pull Requests & Code Reviews

Pull Requests (PRs) are required for merging code into protected branches.

4.1 What is a Pull Request?

A **Pull Request** is a request to merge code from one branch into another (often feature → develop/main).

PRs initiate:

- Code review
- Build validation
- Test execution
- Policy enforcement

4.2 PR Flow

1. Developer completes work in feature/xyz
2. Pushes code
3. Creates PR → Target: develop or main
4. Reviewers comment, approve, or request changes

5. CI pipeline validates
6. Merge completed

4.3 Code Review Best Practices

- Add **minimum 2 reviewers**
- Review **logic, performance, security**
- Validate naming conventions
- Verify unit test coverage
- Avoid reviewing large PRs
- Leave meaningful actionable comments

---

5. Policies, Merge Strategies & Branch Protection

Azure DevOps ensures stable code via branch policies.

5.1 Branch Policies

Navigate:
Project Settings → Repositories → Branches → Select Branch → Branch Policies

Recommended Policies

✓ Require minimum reviewers
✓ Check for linked work items
✓ Enforce comment resolution
✓ Require successful CI build
✓ Limit who can force-push
✓ Prevent direct commits to main/develop
✓ Use "Squash merge" for cleaner history

5.2 Merge Strategies in Azure DevOps

1. Merge Commit

Creates a merge record.

Good For:

- Keeping full branch history
- Large teams

2. Squash Merge (Recommended)

All commits from the feature branch become **a single commit** in main.

Benefits:

- Clean history
- Easier rollback
- Good for CI/CD

3. Rebase and Fast-Forward

Rewrites commit history.

Best For:

- Small teams
- Linear commit history

5.3 Branch Protection

Protection prevents accidental changes.

Protection Capabilities

- Disable force-push
- Block deletion
- Require PR validations
- Role-based permissions

Example: Only Tech Leads can approve PR merges into main.

---

6. LAB – Create Repo, Push Code, Manage Branches (Hands-On)

Perfect for training.

LAB 1: Create a Repository

Steps

1. Go to **Repos → Files**
2. Click **New Repository**
3. Choose:
   - Git
   - Repo name: app-sample
4. Click **Create**

LAB 2: Clone Repository Locally

```
git clone https://dev.azure.com/<org>/<project>/_git/app-sample

cd app-sample
```

LAB 3: Add Code & Push

```
echo "Hello Azure DevOps" > index.html

git add .

git commit -m "Initial commit"

git push origin main
```

LAB 4: Create a Develop & Feature Branch

```
git checkout -b develop

git push origin develop


git checkout -b feature/homepage

git push origin feature/homepage
```

LAB 5: Create a Pull Request

1. Go to **Repos → Pull Requests → New**
2. Source: feature/homepage
3. Target: develop
4. Add Reviewers
5. Add Description
6. Create PR

LAB 6: Add Branch Policies

1. Project Settings → Repositories → Branches
2. Choose main
3. Enable:
   - Minimum reviewers

    - Build validation
    - Comment resolution
    - Limit merge types

LAB 7: Merge the PR

1. Reviewer approves
2. Pipeline passes
3. Choose **Squash Merge**
4. Delete the feature branch after merge

LAB 8: Validate Code in Develop/Main

Check history:

git pull

git log --oneline --graph