2. Source control strategy

└ 2.1 Branching and workflows

└ 2.1.1 Branch strategies: trunk-based, feature, release

---

1. What are the core differences between trunk-based, feature, and release branch strategies?
2. When should you use trunk-based development in a DevOps project?
3. What are the benefits and risks of using long-lived feature branches?
4. How do you implement a release branch strategy in Azure DevOps or GitHub?
5. How do branch strategies impact CI/CD pipeline design?
6. What policies should be applied to branches for quality and security?
7. How do you manage hotfixes in each branch strategy?
8. What is the role of pull requests in trunk-based vs. feature branch workflows?
9. How do you handle versioning and tagging in release branch strategies?
10. What are best practices for merging and resolving conflicts across branch strategies?

---

1. What are the core differences between trunk-based, feature, and release branch strategies?

- Trunk-based uses a single main branch with frequent merges
- Feature branching uses short-lived branches for each feature
- Release branching creates dedicated branches for release preparation and hotfixes

---

2. When should you use trunk-based development in a DevOps project?

Use trunk-based

- when you require rapid delivery,
- frequent integration,
- minimal merge conflicts,
- and a strong focus on CI/CD automation.

---

3. What are the benefits and risks of using long-lived feature branches?

Benefits:

- Isolated development
- Reduced integration risk per feature

Risks:

- Increased merge conflicts
- Stale code
- Delayed integration
- Harder CI/CD automation

---

4. How do you implement a release branch strategy in Azure DevOps or GitHub?

1. Create a release branch from main
2. Stabilize code
3. Apply bug fixes or hotfixes
4. Merge critical changes back to main and other relevant branches
5. Use branch policies for quality gates

5. **How do branch strategies impact CI/CD pipeline design?**

Branch strategies define triggers for builds, tests, and deployments. For example, *main/trunk* triggers production builds; feature branches may trigger only CI builds; release branches may have custom deployment pipelines.

---

6. **What policies should be applied to branches for quality and security?**

Require pull request reviews, enforce status checks, restrict direct pushes, enable branch protection, and mandate code scanning for critical branches.

---

7. **How do you manage hotfixes in each branch strategy?**
    - Trunk-based: commit hotfix directly to main, deploy, then merge with ongoing work.
    - Feature: Apply hotfix to main, rebase feature branches if needed.
    - Release: Apply hotfix to release branch and main, then merge changes as needed.

---

8. **What is the role of pull requests in trunk-based vs. feature branch workflows?**
    - In trunk-based, PRs are typically short-lived and merged quickly.
    - In feature branch workflows, PRs are used to review isolated features and may be open longer, increasing merge complexity.

---

9. **How do you handle versioning and tagging in release branch strategies?**
    - Use tags for releases (e.g., v1.0.0) on release branches;
    - Increment version numbers with each release or hotfix;
    - Track production deployments via tags.

---

10. **What are best practices for merging and resolving conflicts across branch strategies?**
    - Integrate frequently to main
    - Rebase instead of merge where possible
    - Resolve conflicts early
    - Vommunicate changes
    - Automate conflict detection via CI pipelines