3. Build and release pipelines

└ 3.4 Deployment strategy and execution

└ 3.4.4 Hotfix and resiliency planning

---

1. What is a hotfix and when should it be used in a DevOps context?
2. How do you implement a hotfix deployment process in Azure DevOps or GitHub Actions?
3. What strategies enable rapid hotfix deployment with minimal risk?
4. How do you separate hotfix and feature releases in your pipeline?
5. What are the best practices for rolling back a failed hotfix?
6. How do you ensure pipeline and deployment resiliency against failures?
7. What mechanisms can automate detection and recovery from failed deployments?
8. How can you design pipelines to support high availability and quick recovery?
9. What tools and features help with monitoring and alerting for deployment issues?
10. What are key considerations for maintaining compliance and auditability during hotfixes and recovery actions?

---

1. What is a hotfix and when should it be used in a DevOps context?

A hotfix is an urgent, targeted code or configuration change deployed to resolve critical issues in production. Use a hotfix when immediate correction is needed to restore functionality or security.

---

2. How do you implement a hotfix deployment process in Azure DevOps or GitHub Actions?

- Create a hotfix branch from the production or main branch.
- Apply and test the fix.
- Use your CI/CD pipeline to deploy directly to production or a hotfix slot.
- Merge the hotfix back into main and other relevant branches.

---

3. What strategies enable rapid hotfix deployment with minimal risk?

- Maintain a streamlined hotfix pipeline with minimal validation steps.
- Use automated tests focused on affected areas.
- Deploy via slots or canary/rolling deployments for quick rollback.
- Isolate hotfix changes from feature development.

---

4. How do you separate hotfix and feature releases in your pipeline?

- Use branching strategies (e.g., dedicated hotfix branches).
- Tag or label pipeline runs as hotfix or feature.
- Configure CI/CD to allow independent release of hotfixes to production without waiting for feature branch merges.

---

5. What are the best practices for rolling back a failed hotfix?

- Immediately redeploy the previous stable release using pipeline history, slot swap, or rollback tasks.
- Validate system state and data consistency.
- Communicate status and root cause to stakeholders.

6. How do you ensure pipeline and deployment resiliency against failures?

- Use automated health checks after deployment steps.
- Implement retry logic for transient errors.
- Use deployment slots, rolling updates, or blue-green strategies.
- Monitor pipeline health and enforce gates/approvals.

---

7. What mechanisms can automate detection and recovery from failed deployments?

- Configure alerts and automated rollbacks in CI/CD pipelines.
- Use *Azure Monitor* or *Application Insights* to detect failures.
- Employ pipeline conditions to trigger redeployment or revert actions automatically.

---

8. How can you design pipelines to support high availability and quick recovery?

- Use multi-region or multi-zone deployments.
- Keep standby or backup instances updated.
- Automate failover and rollback procedures.
- Test disaster recovery regularly.

---

9. What tools and features help with monitoring and alerting for deployment issues?

- *Azure Monitor*, *Log Analytics*, and *Application Insights* for real-time monitoring and alerts.
- *Service Health* and *Resource Health* for *Azure* infrastructure monitoring.
- *GitHub Actions* and *Azure DevOps* built-in notification and alert integrations.

---

10. What are key considerations for maintaining compliance and auditability during hotfixes and recovery actions?

- Enforce approvals and access controls for hotfix branches and deployments.
- Maintain detailed logs and change history for all hotfixes.
- Document root cause and recovery steps for every incident.