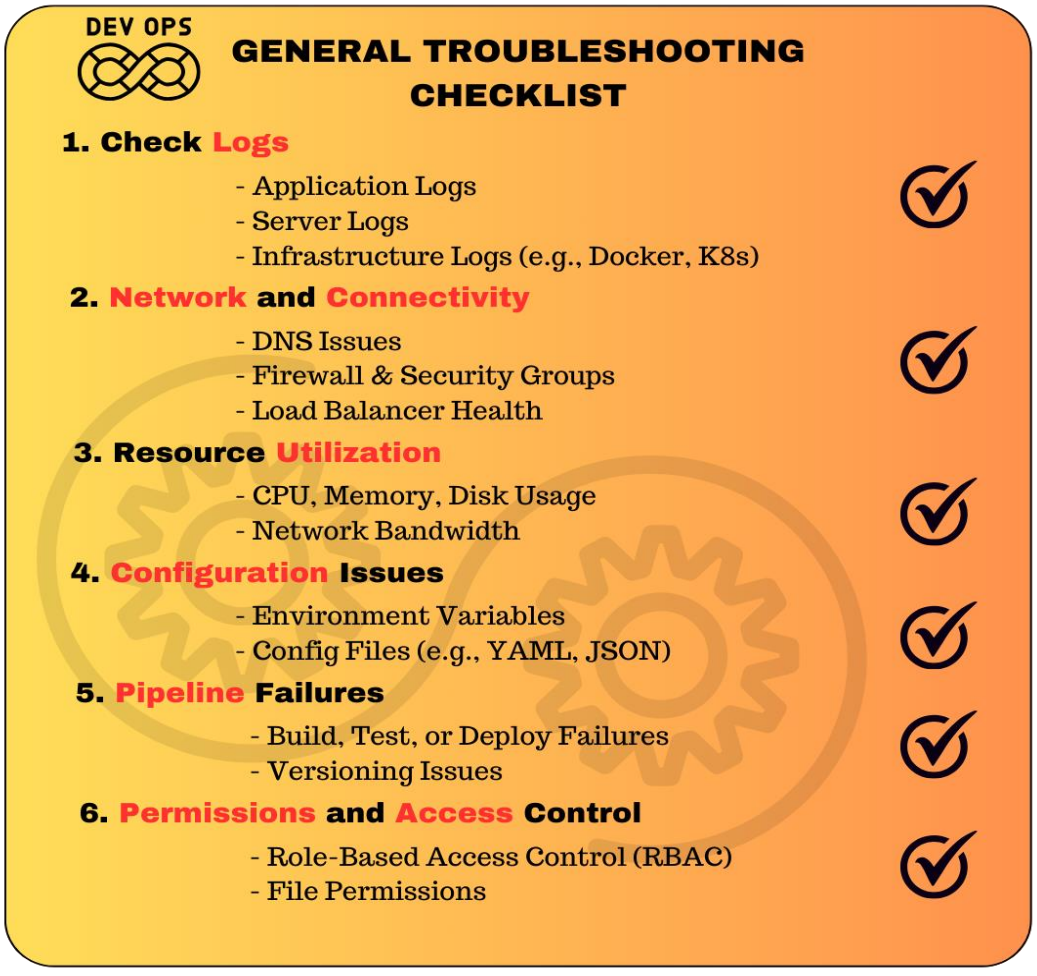
DevOps Shack

General Checklist for Troubleshooting in DevOps

Enroll To Batch-7 DevSecOps & Cloud DevOps Bootcamp

Introduction

In a DevOps environment, troubleshooting is essential for maintaining smooth workflows across Continuous Integration (CI), Continuous Deployment (CD), infrastructure, and applications. With many moving parts, identifying the root



cause of issues can be challenging but crucial for maintaining uptime and efficiency.

This document provides a comprehensive checklist to guide the troubleshooting process by highlighting common areas to investigate when problems arise in DevOps pipelines, infrastructure, and deployments.

Checklist for Troubleshooting in DevOps

Category	What to Check	Common Issues
1. Logs and Error Messages	- Application logs	Misconfigured services, code exceptions, and failures
	- Server logs	Resource limitations, server crashes, and connectivity
	- CI/CD pipeline logs	Pipeline failures, build/test errors
	- Infrastructure logs (e.g., Docker, Kubernetes)	Container crashes, pod failures, orchestration issues
	- Centralized log systems (e.g., ELK, Fluentd)	Log aggregation and quick diagnosis
2. Network and Connectivity	- DNS resolution	DNS misconfiguration, unreachable endpoints
	- Firewall & security group configurations	Blocked ports, disallowed access
	- Network policies in Kubernetes	Network isolation blocking communications between pods
	- Load balancer health	Incorrect routing, failing health checks

Category	What to Check	Common Issues
3. Resource Utilization	- CPU, memory, disk usage	Resource exhaustion, out-of-memory errors
	- Network bandwidth	Congested network slowing down services
	- Monitoring tools (e.g., Prometheus, Grafana)	Alerts on resource thresholds
4. Configuration Issues	- Environment variables	Missing or incorrect configuration variables
	- Configuration files (e.g., YAML, JSON)	Syntax errors, wrong values, incorrect paths
	- Secrets management	Missing/incorrect secrets (e.g., AWS credentials, API keys)
	- Configuration drift between environments	Inconsistent configurations between dev, test, and prod environments
5. Pipeline Failures	- Build, test, or deploy stages	Build failures, test flakiness, deployment misconfigurations
	- Versioning issues in CI/CD pipelines	Dependency version mismatches, outdated libraries
	- Incompatible library or package versions	Breaking changes between library versions
6. Permissions and Access Control	- Role-Based Access Control (RBAC) configurations	Lack of proper permissions for users, services, or applications

Category	What to Check	Common Issues
	- Kubernetes RBAC, AWS IAM policies	Incorrect roles leading to denied access to resources
	- File and directory permissions	Read/write access issues, missing ownership
7. Dependency and Versioning	- Library or package version mismatches	Incompatible dependencies, outdated or unsupported versions
	- System dependencies	Missing libraries, unsupported operating system packages
	- External service compatibility	API version mismatches, service deprecations
8. Security and Certificates	- SSL/TLS certificates	Expired or incorrect certificates, issues with SSL handshake
	- API authentication mechanisms	Incorrect tokens, expired API keys, misconfigured OAuth2 flows
	- Network security policies	Unrestricted inbound/outbound access
9. Infrastructure and Orchestration	- Container orchestration (e.g., Kubernetes)	Pod mis-scheduling, cluster resource constraints, failing services
	- Docker/container configuration	Missing or incorrect environment variables, Dockerfile issues
	- Persistent storage	Volume mounts failing, data loss, incorrect storage class

Category	What to Check	Common Issues
	- Cloud provider resource limits	Hitting quota limits, incorrect scaling configurations
10. Monitoring and Alerts	- Misconfigured monitoring thresholds	Alerts triggering too frequently or not triggering at all
	- Inactive or broken monitoring services	No data ingestion, outdated metrics
	- Lack of detailed metrics for observability	Missing critical metrics for debugging
	- Alert fatigue	Too many alerts without proper categorization or prioritization

Detailed Troubleshooting Areas

1. Logs and Error Messages

Logs provide detailed insights into the behavior of services, applications, and infrastructure. Start by examining logs from the following:

- **Application Logs**: Look for errors, warnings, and anomalies in application behavior.
- **Server Logs**: Identify server-side errors like crashes, resource limitations, or connectivity issues.
- **CI/CD Pipeline Logs**: Review logs for build, test, and deployment failures.
- **Infrastructure Logs**: For issues in containerized environments, check logs of Docker, Kubernetes, or other orchestration tools.
- **Log Aggregation Tools**: Centralized log solutions like Elasticsearch, Fluentd, and Kibana (EFK) or Splunk make it easier to track down issues.

2. Network and Connectivity

Many DevOps issues stem from networking problems. Check:

- **DNS Resolution**: Ensure DNS configurations are correct, and services can resolve domain names.
- **Firewall and Security Groups**: Ensure access is allowed via firewalls, security groups, or network policies.
- **Kubernetes Network Policies**: Check if network policies are blocking traffic between pods.
- **Load Balancer Health**: Ensure the load balancer is routing traffic correctly and performing health checks.

3. Resource Utilization

Insufficient resources can cause services to crash or slow down. Monitor:

- **CPU, Memory, and Disk Usage**: Check for resource exhaustion using monitoring tools like Prometheus, Grafana, or Datadog.
- **Network Bandwidth**: Verify that network usage isn't congesting communication.
- **Scaling**: Check if auto-scaling policies are working as expected in cloud environments.

4. Configuration Issues

Misconfigurations can cause applications to malfunction. Double-check:

- **Environment Variables**: Ensure all required variables are correctly set.
- **Configuration Files (YAML, JSON)**: Look for syntax errors and correct any misconfigurations.

- **Secrets Management**: Ensure credentials and secrets are properly managed and accessible.
- **Configuration Drift**: Validate that the configuration is consistent across development, testing, and production environments.

5. Pipeline Failures

CI/CD pipeline issues are common in DevOps workflows. Investigate:

- **Build, Test, or Deploy Stage Failures**: Identify the exact stage where the pipeline failed and look into logs or error messages.
- **Version Conflicts**: Check for incompatible or outdated versions of libraries and dependencies.
- **Unit/Integration Tests**: Review test logs and verify the reliability of the tests.

6. Permissions and Access Control

Access-related issues can cause service failures or security vulnerabilities. Check:

- **RBAC**: Ensure roles and permissions are correctly configured for users and services.
- **File and Directory Permissions**: Ensure proper ownership and permissions for files used by services.
- **Cloud IAM Policies**: Check if cloud services are authorized correctly using AWS IAM, GCP IAM, or Azure RBAC.

7. Dependency and Versioning Issues

Ensure that all dependencies and external services are properly aligned. Check:

- **Library/Package Versions**: Mismatched or incompatible versions of software dependencies can cause issues.
- **External Service APIs**: Verify that API versions are compatible and not deprecated.

- **System Dependencies**: Check for missing or outdated operating system dependencies.

8. Security and Certificates

Security misconfigurations or expired certificates can lead to service disruptions. Ensure:

- **SSL/TLS Certificates**: Check for expired or incorrectly configured certificates.
- **API Tokens and Authentication**: Verify API tokens, OAuth flows, and authentication mechanisms are properly configured.
- **Network Security Policies**: Ensure security policies are tight but not blocking legitimate access.

9. Infrastructure and Orchestration

For containerized and orchestrated environments, check:

- **Pod Scheduling**: Ensure Kubernetes is scheduling pods correctly and not running into resource constraints.
- **Docker Configuration**: Check Dockerfile configurations and environment variables.
- **Persistent Storage**: Ensure volume mounts and data persistence configurations are correct.
- **Cloud Resource Limits**: Ensure you're not hitting quota or resource limits in your cloud provider.

10. Monitoring and Alerts

Ensure monitoring is configured correctly for better visibility and alerting. Check:

- **Monitoring Thresholds**: Ensure thresholds for metrics like CPU usage, memory usage, and response times are set correctly.
- **Alert Configuration**: Ensure critical alerts are firing and that alert noise is minimized.

- **Detailed Metrics**: Ensure you are capturing detailed enough metrics to diagnose issues.

Conclusion

Effective troubleshooting in DevOps requires a methodical approach to identify issues quickly. This checklist covers the most common areas to examine, from logs and networking to resource utilization and security. By systematically following this checklist, teams can more easily diagnose and resolve issues, keeping services running smoothly and minimizing downtime.