Sample RunBook for Cloud Developer Teams

Table of Contents

Introduction

This RunBook provides detailed procedures for cloud developer teams to handle common issues such as deployment failures, service downtime, performance degradation, security incidents, and cost overruns. Each section includes steps for identification, immediate actions, troubleshooting, and resolution. This document is intended for use by cloud developers, DevOps engineers, and other technical staff responsible for maintaining cloud-based applications and infrastructure.

Deployment Failures

Identification

1. Monitoring Alerts:
    - Check for alerts related to deployment failures in your CI/CD system (e.g., Jenkins, GitLab CI, GitHub Actions).
    - Look for error messages indicating build or deployment failures.
2. User Reports:
    - Users or stakeholders may report issues such as "new features not available," "deployment errors," or "service not updated."

Immediate Actions

1. Acknowledge the Alert:
    - Confirm receipt of the alert in the monitoring or CI/CD system to stop further notifications.
2. Check Deployment Logs:
    - Access the CI/CD system and review the deployment logs for error messages and failed steps.

Troubleshooting Steps

1. Build Logs:
    - Review build logs for any compilation errors, missing dependencies, or failed tests:

      ```
      tail -f /path/to/build/log
      ```

2. Configuration Files:
    - Check configuration files (e.g., Jenkinsfile, gitlab-ci.yml) for errors or recent changes.
3. Environment Issues:
    - Verify that the deployment environment (e.g., staging, production) is correctly configured and accessible.
4. Dependency Checks:
    - Ensure all dependencies (e.g., external libraries, services) are available and compatible with the deployment.

Resolution

1. Fix Build Issues:
    - Correct any errors identified in the build logs. Rebuild the application if necessary.
2. Update Configuration:
    - Correct any issues in the configuration files. Commit and push changes to trigger a new deployment.
3. Environment Setup:
    - Ensure the deployment environment is properly set up. Fix any issues related to environment variables, network configurations, or permissions.
4. Document the Incident:
    - Update the incident ticket with the steps taken and the final resolution.
    - Conduct a post-incident review to identify any preventive measures.

Service Downtime

Identification

1. Monitoring Alerts:
    - Check for alerts related to service downtime in your monitoring system (e.g., CloudWatch, Prometheus).
    - Look for specific error messages such as "Service Unavailable" or "Connection Timeout."
2. User Reports:

- Users may report issues such as "service not reachable," "unable to access application," or "website down."

Immediate Actions

1. Acknowledge the Alert:
   - Confirm receipt of the alert in the monitoring system to stop further notifications.
2. Check Service Status:
   - Verify the status of the affected service using cloud provider dashboards or CLI commands.

Troubleshooting Steps

1. Logs and Metrics:
   - Review service logs and metrics for any errors or anomalies. Use commands such as:

     ```
     tail -f /var/log/my-service/error.log
     ```

   - Check for spikes in CPU, memory, or network usage.
2. Dependency Health:
   - Check the status of any dependent services or resources (e.g., databases, external APIs).
3. Configuration Review:
   - Review configuration files and environment settings for errors or recent changes.
4. Network Diagnostics:
   - Perform network diagnostics to identify connectivity issues (e.g., `ping`, `traceroute`).

Resolution

1. Restart Services:
   - Restart the affected services and verify functionality:

     ```
     sudo systemctl restart my-service
     ```

2. Fix Configuration Issues:
    - Correct any configuration issues identified. Ensure environment variables and network settings are correct.
3. Scale Resources:
    - Increase resources (CPU, memory) for the affected services if necessary. Use auto-scaling features if available.
4. Document the Incident:
    - Update the incident ticket with the steps taken and the final resolution.
    - Conduct a post-incident review to identify any preventive measures.

Performance Degradation

Identification

1. Monitoring Alerts:
    - Check for alerts related to performance degradation in your monitoring system.
    - Look for specific metrics indicating high latency, slow response times, or resource exhaustion.
2. User Reports:
    - Users may report issues such as "application is slow," "pages taking too long to load," or "poor performance."

Immediate Actions

1. Acknowledge the Alert:
    - Confirm receipt of the alert in the monitoring system to stop further notifications.
2. Check System Health:
    - Verify the overall health of the system using cloud provider dashboards or CLI commands.

Troubleshooting Steps

1. Logs and Metrics:
    - Review logs and metrics for any errors or performance bottlenecks. Use commands such as:

      ```
      tail -f /var/log/my-service/performance.log
      ```

    - Check for resource utilization patterns (CPU, memory, disk I/O).
2. Application Profiling:
    - Use profiling tools to identify performance bottlenecks in the application code (e.g., `perf`, `py-spy` for Python applications).
3. Dependency Health:
    - Check the performance and health of dependent services or resources (e.g., databases, external APIs).
4. Configuration Review:
    - Review configuration files and environment settings for optimization opportunities.

Resolution

1. Optimize Code:
    - Identify and optimize inefficient code paths. Refactor code to improve performance and reduce resource usage.
2. Scale Resources:
    - Increase resources (CPU, memory) for the affected services if necessary. Use auto-scaling features if available.
3. Update Configuration:
    - Adjust configuration settings to optimize performance (e.g., connection pools, caching strategies).
4. Document the Incident:
    - Update the incident ticket with the steps taken and the final resolution.
    - Conduct a post-incident review to identify any preventive measures.

Security Incidents

Identification

1. Monitoring Alerts:
    - Check for alerts related to security incidents in your security monitoring system (e.g., AWS GuardDuty, Azure Security Center).
    - Look for specific error messages or alerts indicating unauthorized access, data breaches, or malware detection.
2. User Reports:
    - Users may report suspicious activities such as "unauthorized access," "data breaches," or "unusual activity."

Immediate Actions

1. Acknowledge the Alert:
    - Confirm receipt of the alert in the security monitoring system to stop further notifications.
2. Isolate Affected Systems:
    - Isolate affected systems to prevent further unauthorized access or damage. Use network segmentation or disable compromised accounts.

Troubleshooting Steps

1. Log Review:
    - Review security logs for any suspicious activity or unauthorized access attempts. Use commands such as:

      ```
      tail -f /var/log/auth.log
      ```

    - Look for unusual login attempts, failed login attempts, or unexpected changes.
2. Vulnerability Assessment:
    - Perform a vulnerability assessment to identify security weaknesses. Use tools like `nmap` or `OpenVAS`.
3. Access Control Review:
    - Review access control settings and permissions for any anomalies or misconfigurations.
4. Incident Analysis:
    - Analyze the incident to understand the attack vector, scope, and impact. Identify the root cause of the security breach.

Resolution

1. Patch Vulnerabilities:
    - Apply patches or updates to fix identified vulnerabilities. Ensure all systems are up to date.
2. Revoke Access:
    - Revoke access for compromised accounts and issue new credentials. Strengthen access control policies.
3. Strengthen Security Measures:
    - Implement additional security measures such as multi-factor authentication (MFA), intrusion detection systems (IDS), and regular security audits.
4. Document the Incident:
    - Update the incident ticket with the steps taken and the final resolution.
    - Conduct a post-incident review to identify any preventive measures.

Cost Overruns

Identification

1. Monitoring Alerts:
    - Check for alerts related to cost overruns in your cloud cost management system (e.g., AWS Cost Explorer, Azure Cost Management).
    - Look for specific alerts indicating budget breaches or unexpected cost spikes.
2. Financial Reports:
    - Review financial reports and billing statements for any unusual or unexpected charges.

Immediate Actions

1. Acknowledge the Alert:
    - Confirm receipt of the alert in the cost management system to stop further notifications.

2. Check Usage Reports:
    - Review detailed usage reports to identify services or resources contributing to the cost overrun.

Troubleshooting Steps

1. Resource Utilization:
    - Check resource utilization to identify any over-provisioned or underutilized resources. Use cloud provider dashboards or CLI commands.
2. Cost Breakdown:
    - Analyze the cost breakdown to identify the most expensive services or resources. Look for unexpected spikes in usage or costs.
3. Configuration Review:
    - Review configuration settings for cost optimization opportunities. Ensure resources are configured to scale appropriately.
4. Usage Patterns:
    - Identify any unusual usage patterns or activities that could contribute to cost overruns.

Resolution

1. Optimize Resources:
    - Optimize or resize over-provisioned resources. Use reserved instances or savings plans to reduce costs.
2. Implement Cost Controls:
    - Implement cost controls such as budgets, spending limits, and automated alerts to monitor and manage costs.
3. Update Configuration:
    - Adjust configuration settings to optimize cost (e.g., auto-scaling policies, instance types).
4. Document the Incident:
    - Update the incident ticket with the steps taken and the final resolution.
    - Conduct a post-incident review to identify any preventive measures.