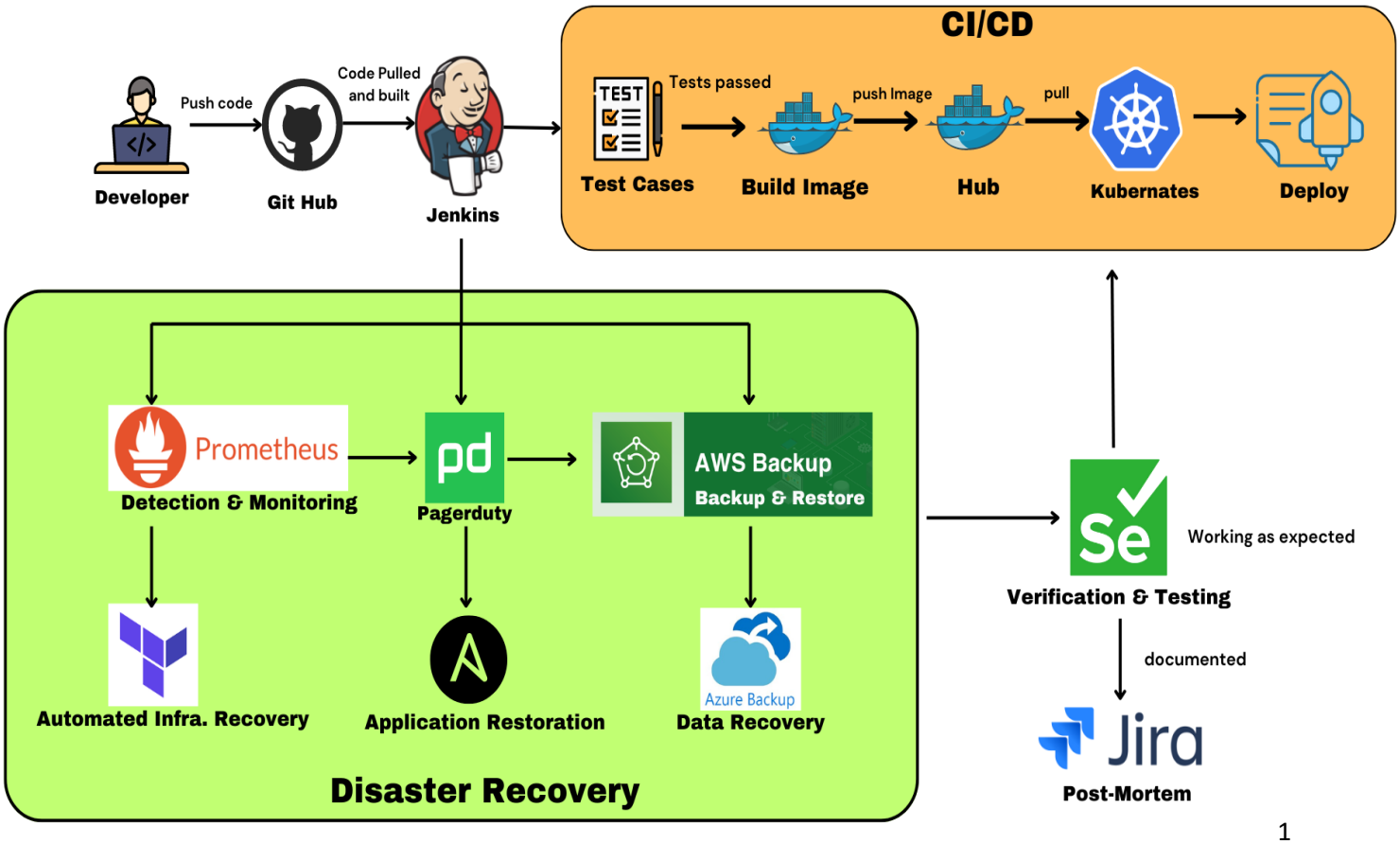
DevOps Shack

Integrating Disaster Recovery into the CI/CD Pipeline

Introduction

In modern DevOps practices, Continuous Integration and Continuous Deployment (CI/CD) pipelines enable rapid development and deployment cycles. However, integrating **Disaster Recovery (DR)** strategies within these pipelines ensures high



availability, resilience, and quick recovery from system failures. In this document, we will cover how to implement disaster recovery within a CI/CD pipeline, why it is necessary, and the tools involved in each stage of the implementation.

Why Integrate Disaster Recovery with CI/CD?

Incorporating Disaster Recovery strategies into a CI/CD process is crucial for several reasons:

- **Minimizing Downtime**: Automatic detection and recovery minimize system downtime during failures.
- **Maintaining Service Continuity**: Ensures that service availability is preserved even during infrastructure failures.
- **Ensuring Data Integrity**: DR plans help in restoring critical data, preventing data loss or corruption during incidents.
- **Automating Recovery Processes**: Automated disaster recovery reduces manual intervention, speeding up the recovery process.
- **Comprehensive Testing**: DR integrated with CI/CD allows for regular testing of recovery plans, ensuring the system is always ready to handle incidents.

Step-by-Step Implementation

This implementation will utilize tools at each stage to integrate DR into your CI/CD pipeline. Here's a breakdown of the process:

1. Code Commit & Version Control (Git)

Why?

A robust version control system is the foundation of any CI/CD pipeline. It ensures that all code changes are tracked, documented, and can be easily reverted in case of errors.

How?

- Set up a Git repository (GitHub/GitLab) to manage the codebase.
- Implement branch management policies to ensure that changes to the main branch are stable.
- Configure webhooks to trigger CI jobs when new commits are pushed.

Tool: Git

```
git init
git remote add origin <repository-url>
git add .
git commit -m "Initial Commit"
git push origin main
```

2. Continuous Integration (Jenkins)

Why?

Continuous Integration tools like Jenkins automate the process of testing and building code after every change, ensuring that any code introduced into the pipeline is functional and doesn't break the existing workflow.

How?

- Set up a Jenkins server.
- Create a Jenkins job that triggers on code changes.
- Add build steps to compile and run unit tests.

Tool: Jenkins

```
pipeline {
  agent any
  stages {
    stage('Build') {
```

```
        steps {
            sh 'mvn clean install'
        }
    }
    stage('Test') {
        steps {
            sh 'mvn test'
        }
    }
  }
}
```

Add yours according to tools .

3. Deployment & Orchestration (Kubernetes)

Why?

Using Kubernetes for deployment ensures scalability and high availability. It allows you to manage containerized applications, enabling easy recovery and failover mechanisms.

How?

- Set up a Kubernetes cluster (on-premise or cloud-based such as EKS, GKE, or AKS).
- Configure Helm charts to manage deployment configuration.
- Use Kubernetes' native features like ReplicaSets and StatefulSets to ensure redundancy.

Tool: Kubernetes

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp-container
        image: myapp:latest
        ports:
        - containerPort: 80
```

4. Monitoring and Detection (Prometheus)

Why?

Monitoring systems like Prometheus are crucial for detecting issues in real-time. They gather metrics from your application and infrastructure, alerting you to any anomalies or failures.

How?

- Install Prometheus and configure it to scrape metrics from Kubernetes nodes.
- Set up alert rules for critical metrics, such as high CPU usage or application unavailability.
- Integrate with a notification system (e.g., PagerDuty) for incident alerts.

Tool: Prometheus

```
scrape_configs:
 - job_name: 'kubernetes-nodes'
   kubernetes_sd_configs:
    - role: node
   relabel_configs:
    - source_labels: [__address__]
      regex: '(.*):10255'
      target_label: __address__
      replacement: '$1:9100'
```

5. Incident Response (PagerDuty)

Why?

PagerDuty provides real-time alerting and on-call management, ensuring that your team is promptly notified of any critical incidents.

How?

- Create an account on PagerDuty and configure a service for your application.
- Integrate Prometheus with PagerDuty using a webhook or the Alertmanager configuration.

- Set up on-call schedules and escalation policies.

Tool: PagerDuty

```
route:
 receiver: 'pagerduty'
receivers:
 - name: 'pagerduty'
   pagerduty_configs:
    - service_key: 'your-service-key'
```

6. Backup & Restore (AWS Backup)

Why?

Backups ensure that your data can be restored in the event of a disaster. AWS Backup provides automated backups for databases, file systems, and applications running on AWS.

How?

- Create an AWS Backup plan to take regular snapshots of your infrastructure.
- Configure lifecycle rules to retain backups for long-term storage.
- Use these backups during a disaster event to restore services.

Tool: AWS Backup

```
aws backup create-backup-plan --backup-plan "{
 \"BackupPlanName\": \"myBackupPlan\",
 \"Rules\": [{\"RuleName\": \"daily-backup\",
 \"TargetBackupVaultName\": \"myBackupVault\",
 \"ScheduleExpression\": \"cron(0 12 * * ? *)\",
```

```
 \"Lifecycle\": {\"DeleteAfterDays\": 30}}]}"
```

7. Automated Infrastructure Recovery (Terraform)

Why?

Terraform automates the provisioning of cloud infrastructure, ensuring that your environment can be quickly rebuilt in case of a disaster.

How?

- Write Terraform configuration files to define your infrastructure (networks, VMs, storage).
- Store Terraform state in a remote backend (e.g., S3).
- Use Terraform to re-provision the infrastructure when a disaster strikes.

Tool: Terraform

```
provider "aws" {
 region = "us-east-1"
}

resource "aws_instance" "web" {
 ami           = "ami-0c55b159cbfafe1f0"
 instance_type = "t2.micro"
}
```

8. Application Restoration (Ansible)

Why?

Ansible automates the deployment of applications, ensuring that restored infrastructure can have services running as quickly as possible.

How?

- Write Ansible playbooks to define tasks for restoring application services.
- Use Ansible to re-deploy services onto restored infrastructure.

Tool: Ansible

```
---
- hosts: all
  tasks:
    - name: Install Nginx
      apt: name=nginx state=latest
    - name: Start Nginx
      service:
        name: nginx
        state: started
```

9. Data Recovery (Azure Backup)

Why?

Azure Backup offers secure backup solutions for VMs, databases, and files stored in the cloud. It ensures that critical data can be restored in case of data loss.

How?

- Set up Azure Backup to take periodic snapshots of data.
- Configure restore points and retention policies.
- Use Azure's restore capabilities to recover databases and files.

Tool: Azure Backup

```
az backup vault create --name MyVault --resource-group MyResourceGroup --location eastus
```

```
az backup protection enable-for-vm --resource-group MyResourceGroup --vault-name MyVault --vm MyVM
```

10. Verification & Testing (Selenium)

Why?

After recovery, it's crucial to verify that the application is functioning as expected. Selenium can be used to automate this process by running tests on the restored application.

How?

- Set up Selenium to run automated tests against your application.
- Write test scripts to validate the functionality of critical components.

Tool: Selenium

```
from selenium import webdriver


driver = webdriver.Chrome()
driver.get("http://myapp-url.com")
assert "MyApp" in driver.title
driver.quit()
```

11. Post-Mortem Analysis (JIRA, Confluence)

Why?

After an incident is resolved, it's essential to document the root cause and the actions taken. This helps to improve future disaster recovery processes.

How?

- Use JIRA to create incident tickets and track issues.
- Document the incident response process and future action items in Confluence.

Tool: JIRA / Confluence

Conclusion

By integrating Disaster Recovery into your CI/CD pipeline, you ensure that your system remains resilient, highly available, and capable of quickly recovering from disasters. Each tool plays a specific role in managing the disaster recovery process, ensuring a seamless integration with the CI/CD pipeline. Following this step-by-step implementation will safeguard your application against unexpected failures and minimize downtime, ensuring business continuity.