Arkitektur som kod

Infrastructure as Code i praktiken

Kodarkitektur Bokverkstad

Contents

Chapter 1

Inledning till arkitektur som kod

Arkitektur som kod (Architecture as Code) representerar ett paradigmskifte inom systemutveckling där hela systemarkitekturen definieras, versionshanteras och hanteras genom kod. Detta approach möjliggör samma metodiker som traditionell mjukvaruutveckling för hela organisationens tekniska landskap.

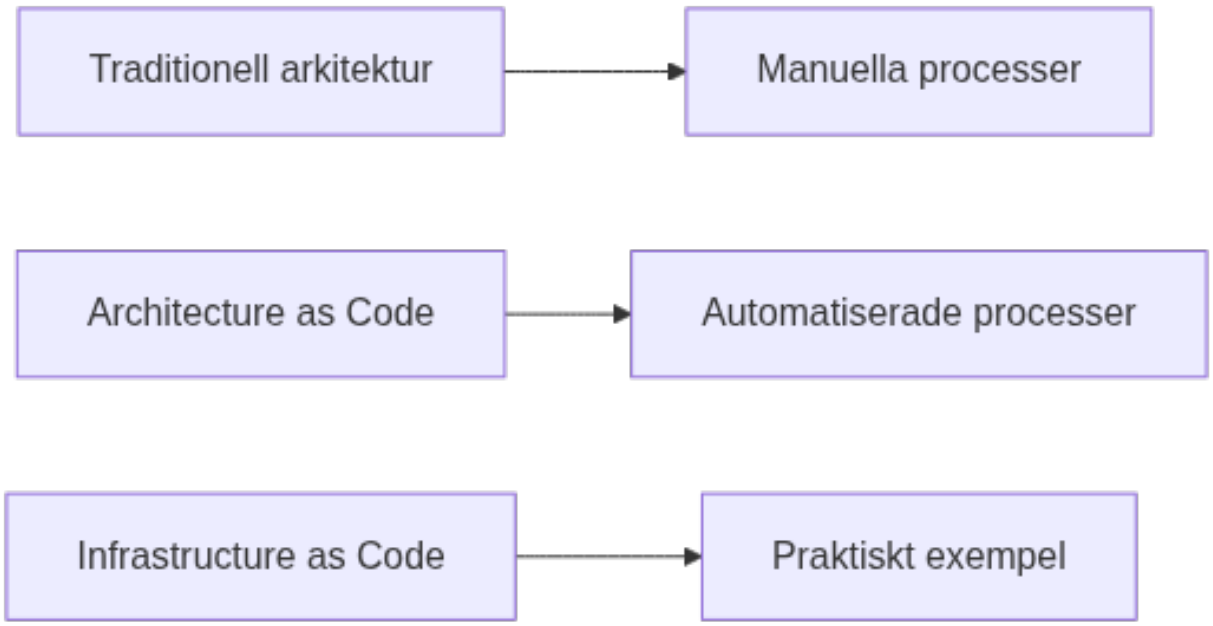

Figure 1.1: Inledning till arkitektur som kod

Diagrammet illustrerar evolutionen från manuella processer till den omfattande visionen av Architecture as Code, där hela systemarkitekturen kodifieras.

1.1 Evolution mot arkitektur som kod

Traditionella metoder för systemarkitektur har ofta varit manuella och dokumentbaserade. Architecture as Code bygger på etablerade principer från mjukvaruutveckling och tillämpar dessa på hela systemlandskapet.

Detta inkluderar inte bara infrastrukturkomponenter, utan även applikationsarkitektur, dataflöden, säkerhetspolicies, compliance-regler och organisatoriska strukturer - allt definierat som kod.

1.2 Definition och omfattning

Architecture as Code definieras som praktiken att beskriva, versionhantera och automatisera hela systemarkitekturen genom maskinläsbar kod. Detta omfattar applikationskomponenter, integrationsmönster, dataarkitektur, infrastruktur och organisatoriska processer.

Denna holistiska approach möjliggör end-to-end automatisering där förändringar i krav automatiskt propagerar genom hela arkitekturen - från applikationslogik till deployment och monitering.

1.3 Bokens syfte och målgrupp

Denna bok vänder sig till systemarkitekter, utvecklare, projektledare och IT-beslutsfattare som vill förstå och implementera Architecture as Code i sina organisationer.

Läsaren kommer att få omfattande kunskap om hur hela systemarkitekturen kan kodifieras, från grundläggande principer till avancerade arkitekturmönster som omfattar hela organisationens digitala ekosystem.

Källor: - ThoughtWorks. “Architecture as Code: The Next Evolution.” Technology Radar, 2024. - Martin, R. “Clean Architecture: A Craftsman’s Guide to Software Structure.” Prentice Hall, 2017.

Chapter 2

Grundläggande principer för Architecture as Code

Architecture as Code bygger på fundamentala principer som säkerställer framgångsrik implementation av kodifierad systemarkitektur. Dessa principer omfattar hela systemlandskapet och skapar en helhetssyn för arkitekturhantering.

Figure 2.1: Grundläggande principer diagram

Diagrammet visar det naturliga flödet från deklarativ kod genom versionskontroll och automatisering till reproducerbarhet och skalbarhet - de fem grundpelarna inom Architecture as Code.

2.1 Deklarativ arkitekturdefinition

Den deklarativa approachen inom Architecture as Code innebär att beskriva önskat systemtillstånd på alla nivåer - från applikationskomponenter till infrastruktur. Detta skiljer sig från imperativ programmering där varje steg måste specificeras explicit.

Deklarativ definition möjliggör att beskriva arkitekturens önskade tillstånd, vilket Architecture as Code utvidgar till att omfatta applikationsarkitektur, API-kontrakt och organisatoriska strukturer.

2.2 Helhetsperspektiv på kodifiering

Architecture as Code omfattar hela systemekosystemet genom en holistisk approach. Detta inkluderar applikationslogik, dataflöden, säkerhetspolicies, compliance-regler och organisationsstrukturer.

Ett praktiskt exempel är hur en förändring i en applikations API automatiskt kan propagera genom

hela arkitekturen - från säkerhetskonfigurationer till dokumentation - allt eftersom det är definierat som kod.

2.3 Immutable architecture patterns

Principen om immutable arkitektur innebär att hela systemarkitekturen hanteras genom oföränderliga komponenter. Istället för att modifiera befintliga delar skapas nya versioner som ersätter gamla på alla nivåer.

Detta skapar förutsägbarhet och eliminerar architectural drift - där system gradvis divergerar från sin avsedda design över tid.

2.4 Testbarhet på arkitekturnivå

Architecture as Code möjliggör testning av hela systemarkitekturen, inte bara enskilda komponenter. Detta inkluderar validering av arkitekturmönster, compliance med designprinciper och verifiering av end-to-end-flöden.

Arkitekturtester validerar designbeslut, systemkomplexitet och säkerställer att hela arkitekturen fungerar som avsett.

2.5 Documentation as Code

Documentation as Code (DaC) representerar principen att behandla dokumentation som en integrerad del av kodbasen snarare än som ett separat artefakt. Detta innebär att dokumentation lagras tillsammans med koden, versionshanteras med samma verktyg och genomgår samma kvalitetssäkringsprocesser som applikationskoden.

2.5.1 Fördelar med Documentation as Code

Versionskontroll och historik: Genom att lagra dokumentation i Git eller andra versionskontrollsystem får organisationer automatisk spårbarhet av förändringar, möjlighet att återställa tidigare versioner och full historik över dokumentationens utveckling.

Kollaboration och granskning: Pull requests och merge-processer säkerställer att dokumentationsändringar granskas innan de publiceras. Detta förbättrar kvaliteten och minskar risken för felaktig eller föråldrad information.

CI/CD-integration: Automatiserade pipelines kan generera, validera och publicera dokumentation automatiskt när kod förändras. Detta eliminerar manuella steg och säkerställer att dokumentationen alltid är uppdaterad.

2.5.2 Praktisk implementation

```yaml
# .github/workflows/docs.yml
name: Documentation Build and Deploy
on:
  push:
    paths: ['docs/**', 'README.md']
  pull_request:
    paths: ['docs/**']

jobs:
  build-docs:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '18'

      - name: Install dependencies
        run: npm install

      - name: Generate documentation
        run: |
          npm run docs:build
          npm run docs:lint

      - name: Deploy to GitHub Pages
        if: github.ref == 'refs/heads/main'
        uses: peaceiris/actions-gh-pages@v3
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./docs/dist
```

Moderna verktyg som GitBook, Gitiles och MkDocs möjliggör automatisk generering av webbdokumentation från Markdown-filer lagrade tillsammans med koden.

2.6 Requirements as Code

Requirements as Code (RaC) transformerar traditionell kravspecifikation från textdokument till maskinläsbar kod som kan exekveras, valideras och automatiseras. Detta paradigmskifte möjliggör kontinuerlig verifiering av att systemet uppfyller sina krav genom hela utvecklingslivscykeln.

2.6.1 Automatisering och traceability

Automatiserad validering: Krav uttryckta som kod kan exekveras automatiskt mot systemet för att verifiera compliance. Detta eliminerar manuell testning och säkerställer konsekvent validering.

Direkt koppling mellan krav och kod: Varje systemkomponent kan kopplas tillbaka till specifika krav, vilket skapar fullständig traceability från affärsbehov till teknisk implementation.

Continuous compliance: Förändringar i systemet valideras automatiskt mot alla definierade krav, vilket förhindrar regression och säkerställer ongoing compliance.

2.6.2 Praktiskt exempel med Open Policy Agent (OPA)

```yaml
# requirements/security-requirements.yaml
apiVersion: policy/v1
kind: RequirementSet
metadata:
  name: svenska-sakerhetskrav
  version: "1.2"
spec:
  requirements:
    - id: SEC-001
      type: security
      description: "Alla S3 buckets måste ha kryptering aktiverad"
      priority: critical
      compliance: ["GDPR", "ISO27001"]
      policy: |
        package security.s3_encryption

        deny[msg] {
          input.resource_type == "aws_s3_bucket"
          not input.server_side_encryption_configuration
          msg := "S3 bucket måste ha server-side encryption"
        }

    - id: GDPR-001
      type: compliance
```

```
    description: "Persondata måste lagras inom EU/EES"
    priority: critical
    compliance: ["GDPR"]
    policy: |
      package compliance.data_residency

      deny[msg] {
        input.resource_type == "aws_rds_instance"
        not contains(input.availability_zone, "eu-")
        msg := "RDS instans måste placeras i EU-region"
      }
```

2.6.3 Validering och test-automation

Requirements as Code integreras naturligt med test-automation genom att krav blir executable specifications:

```python
# test/requirements_validation.py
import yaml
import opa

class RequirementsValidator:
    def __init__(self, requirements_file: str):
        with open(requirements_file, 'r') as f:
            self.requirements = yaml.safe_load(f)

    def validate_requirement(self, req_id: str, system_config: dict):
        requirement = self.find_requirement(req_id)
        policy_result = opa.evaluate(
            requirement['policy'],
            system_config
        )
        return {
            'requirement_id': req_id,
            'status': 'passed' if not policy_result else 'failed',
            'violations': policy_result
        }

    def validate_all_requirements(self) -> dict:
        results = []
        for req in self.requirements['spec']['requirements']:
```

```
            result = self.validate_requirement(req['id'], self.system_config)
            results.append(result)

        return {
            'total_requirements': len(self.requirements['spec']['requirements']),
            'passed': len([r for r in results if r['status'] == 'passed']),
            'failed': len([r for r in results if r['status'] == 'failed']),
            'details': results
        }
```

Svenska organisationer drar särskild nytta av Requirements as Code för att automatiskt validera GDPR-compliance, finansiella regleringar och myndighetskrav som konstant måste uppfyllas.

Källor: - Red Hat. “Architecture as Code Principles and Best Practices.” Red Hat Developer. - Martin, R. “Clean Architecture: A Craftsman’s Guide to Software Structure.” Prentice Hall, 2017. - ThoughtWorks. “Architecture as Code: The Next Evolution.” Technology Radar, 2024. - GitLab. “Documentation as Code: Best Practices and Implementation.” GitLab Documentation, 2024. - Open Policy Agent. “Policy as Code: Expressing Requirements as Code.” CNCF OPA Project, 2024. - Atlassian. “Documentation as Code: Treating Docs as a First-Class Citizen.” Atlassian Developer, 2023. - NIST. “Requirements Engineering for Secure Systems.” NIST Special Publication 800-160, 2023.

Chapter 3

Versionhantering och kodstruktur

Effektiv versionhantering utgör ryggraden i Infrastructure as Code-implementationer. Genom att tillämpa samma metoder som mjukvaruutveckling på infrastrukturdefinitioner skapas spårbarhet, samarbetsmöjligheter och kvalitetskontroll.

Figure 3.1: Versionhantering och kodstruktur

Diagrammet illustrerar det typiska flödet från Git repository genom branching strategy och code review till slutlig deployment, vilket säkerställer kontrollerad och spårbar infrastrukturutveckling.

3.1 Git-baserad arbetsflöde för infrastruktur

Git utgör standarden för versionhantering av IaC-kod och möjliggör distribuerat samarbete mellan team-medlemmar. Varje förändring dokumenteras med commit-meddelanden som beskriver vad som ändrats och varför, vilket skapar en komplett historik över infrastrukturutvecklingen.

3.2 Kodorganisation och modulstruktur

Välorganiserad kodstruktur är avgörande för maintainability och collaboration i större IaC-projekt. Modulär design möjliggör återanvändning av infrastrukturkomponenter across olika projekt och miljöer.

Källor: - Atlassian. “Git Workflows for Infrastructure as Code.” Atlassian Git Documentation.