International Conference on Science, Engineering & Management Trends

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

Dependency Mapping and Control-Plane Isolation for Resilient Distributed Cloud Architectures

[1]*Raghvendra Rao, [2]* Dr. Yashpal Joshi

[1]*Phd. Research Scholar, Department of Computer Science and Engineering, Saveetha Amarawati University, Vijayawada, Andhra Pradesh, India

[2]*PhD Guide, Department of Computer Science and Engineering, Saveetha Amarawati University, Vijayawada, Andhra Pradesh, India

Abstract- Distributed cloud services often appear redundant at the component level while sharing hidden dependencies on identity, configuration, secrets and deployment pipelines. This paper proposes a dependency-mapping method that starts from user journeys, annotates every dependency with its failure behaviour—mandatory, optional, delayed through expiry, or needed only for recovery—and separates the data plane that serves requests from the control plane that changes and restores the system. Applied to an illustrative service with 13 components and four journeys, the method shows that the identity service blocks all four journeys immediately, while configuration and secrets services block only one journey at first but three after cached leases and credentials expire. An expiry-horizon analysis shows why short outage tests miss these effects: with assumed horizons of 1, 4, 12 and 72 hours, a one-hour control-plane test reveals only the first. The method gives architects a practical way to locate shared risks and to plan continuity tests that last long enough to cross the relevant expiry horizons.

Keywords- Distributed cloud, dependency graph, control plane, data plane, regional isolation, static stability, common-cause failure, resilience testing

I. INTRODUCTION

Cloud services are built from many managed and self-operated components [1]. Spreading them across zones, regions and providers is intended to isolate failures, yet outage studies show that incidents frequently arise from shared dependencies, configuration changes and hidden coupling that defeat apparent redundancy [2]. Dependability depends on how the complete system behaves when a part fails, not on the reliability of each part in isolation [3].

A common gap is that architecture diagrams show where components run but not what happens when a link fails. Some links block every request immediately; others matter only after a certificate, token or lease expires; others are needed only during recovery. This paper contributes: (1) a journey-centred dependency-mapping method with annotated failure behaviour; (2) an impact analysis that identifies shared nodes on many journeys; and (3) an expiry-horizon analysis that links control-plane dependencies to the duration of continuity tests.

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

Background

Microservice architectures increase the number of service-to-service interactions and therefore the number of dependencies that must be understood [4]. Zero-trust designs evaluate identity and policy on every request, which strengthens security but makes the identity and policy services central to availability [5]. Partial failures, in which components appear healthy while users experience errors, further complicate diagnosis [6]. Cyber-resilience guidance therefore recommends understanding dependencies so that systems can withstand and recover from adverse conditions [7].

II. DEPENDENCY-MAPPING METHOD

1. Journeys and Annotated Links

Mapping begins with a small number of important user journeys, not with an enterprise-wide inventory. For each journey, the components needed to complete it are recorded, and every link is classified by failure behaviour (Table I). Fig. 1 shows an illustrative service with four journeys: browsing, placing an order, deploying a fix and restoring data.

Table 1: Classification of Dependency Links

Link type	Effect when the dependency fails	Example
Mandatory	Journey fails immediately	Order service calls payment adapter
Optional	Journey continues with reduced quality	Catalogue reads without cache
Delayed	Journey fails after a lease, token or certificate expires	Gateway uses cached routing configuration
Recovery only	No effect until restoration is needed	Database relies on backup store

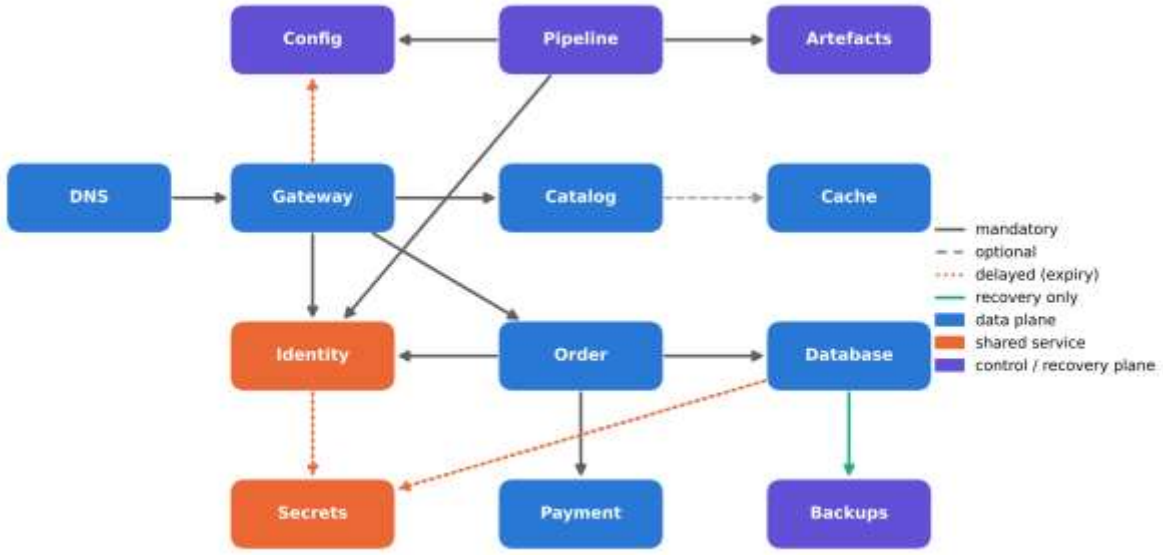


Fig. 1. Illustrative dependency graph for a distributed service. Links are annotated as mandatory, optional, delayed through expiry or needed only for recovery; nodes are coloured by plane.

2. Data and Control Planes

The data plane handles ordinary requests, while the control plane changes configuration, placement and policy and supports deployment and recovery. A control-plane failure need not stop established traffic, but it can prevent scaling, certificate renewal and recovery. A design is statically stable when the data plane can keep serving using locally held configuration and credentials while the control plane is

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

unavailable. The dependency map should therefore record, for each delayed link, how long the data plane can continue before the held state expires.

III. ANALYSIS

1. Journey Impact of Each Component

For each component, the number of journeys blocked immediately and after expiry is counted (Fig. 2). In the illustrative service, the identity service blocks all four journeys immediately, including the deploy and restore journeys needed to repair it, so its failure can also prevent recovery. The configuration and secrets services each block one journey immediately but three once cached state expires, so a short test would underestimate their importance. The cache blocks no journey, because its link is optional.

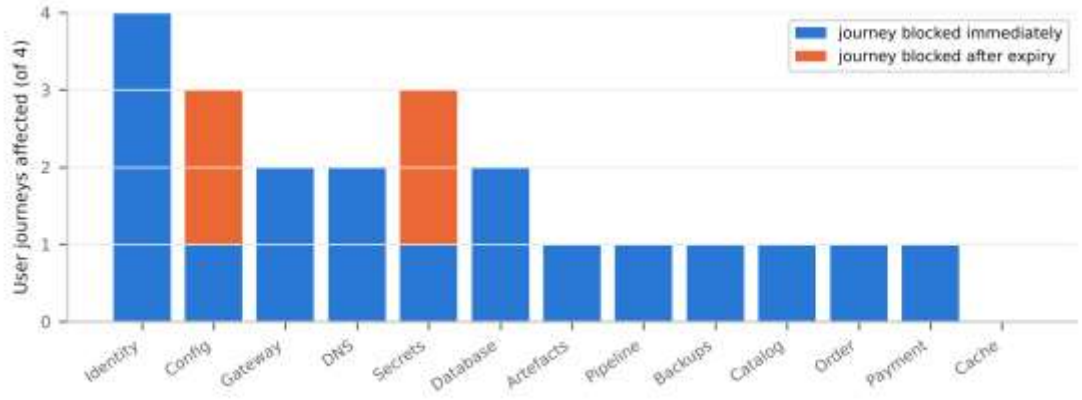


Fig. 2. Number of the four journeys blocked by the failure of each component, immediately and after cached leases, tokens or certificates expire. Illustrative service of Fig. 1.

2. Expiry Horizons and Test Duration

Fig. 3 shows assumed expiry horizons for state held by the data plane. If the control plane is unavailable, cached authorisation decisions expire after 1 hour, service tokens after 4, configuration leases after 12, and certificates inside their renewal window after 72 hours. A one-hour control-plane outage test would reveal only the first effect. Continuity tests must either run long enough to cross the relevant horizon or state explicitly that they have not done so. Fault-injection practice can make such tests routine [8].

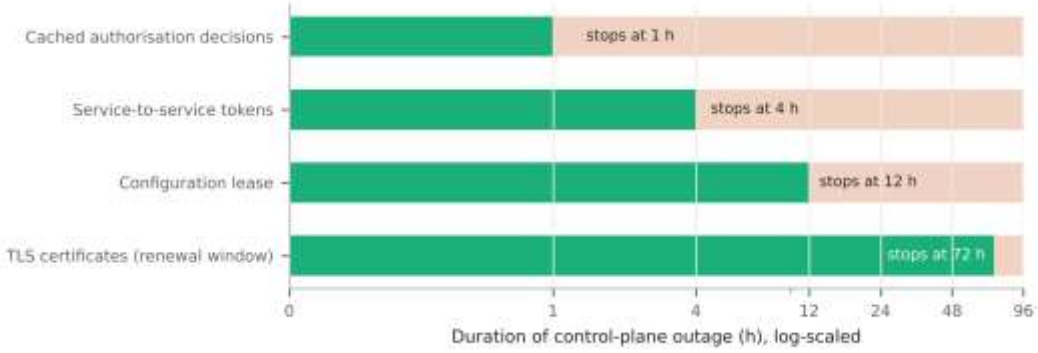


Fig. 3. Assumed expiry horizons: how long data-plane functions continue during a control-plane outage before held state expires. Illustrative values.

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

IV. DESIGN IMPLICATIONS

Table II summarises the design responses suggested by the analysis. Each response reduces either the number of journeys that share a component or the speed at which a control-plane failure reaches the data plane.

Table 2: Design Responses to Dependency Findings

Finding	Design response	Evidence to collect
Component on every journey	Regional replicas; local token validation	Journey success with component removed
Recovery depends on failed component	Independent recovery identity and path	Restore test without production identity
Delayed dependency	Longer-lived held state; renewal margins	Test that crosses the expiry horizon
Regions share automation	Staged, region-by-region changes	Change records per region

Regions are useful isolation units only when their remaining shared dependencies—account administration, global routing, secrets and deployment automation—are understood, and the map makes these visible [2].

V. CONCLUSION

This paper proposed a journey-centred dependency-mapping method that annotates links by failure behaviour and separates data and control planes. In an illustrative service, it identified the identity service as a dependency of every journey and showed that configuration and secrets services triple their impact once cached state expires. Future work will apply the method to production services, measure actual expiry horizons and validate the maps with long-duration control-plane outage tests.

REFERENCES

1. P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST SP 800-145, 2011, doi: 10.6028/NIST.SP.800-145.
2. H. S. Gunawi *et al.*, "Why does the cloud stop computing? Lessons from hundreds of service outages," in *Proc. 7th ACM Symp. Cloud Comput. (SoCC)*, 2016, pp. 1–16, doi: 10.1145/2987550.2987583.
3. A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, 2004, doi: 10.1109/TDSC.2004.2.
4. R. Chandramouli, "Security strategies for microservices-based application systems," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST SP 800-204, 2019, doi: 10.6028/NIST.SP.800-204.
5. S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero trust architecture," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST SP 800-207, 2020, doi: 10.6028/NIST.SP.800-207.
6. P. Huang *et al.*, "Gray failure: The Achilles' heel of cloud-scale systems," in *Proc. 16th Workshop Hot Topics Oper. Syst. (HotOS)*, 2017, pp. 150–155, doi: 10.1145/3102980.3103005.
7. R. Ross *et al.*, "Developing cyber-resilient systems: A systems security engineering approach," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST SP 800-160 Vol. 2 Rev. 1, 2021, doi: 10.6028/NIST.SP.800-160v2r1.

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

8. A. Basiri *et al.*, "Chaos engineering," *IEEE Softw.*, vol. 33, no. 3, pp. 35–41, 2016, doi: 10.1109/MS.2016.60.