App-Bisect
Autonomous Healing for μService-based Apps

Shriram Rajagopalan
Hani Jamjoom
IBM T. J. Watson Research
From Monoliths to Microservices

Service instances

Monolithic Service

Functionalities

Microservice instances

Well-defined API

Microservices
Each functionality in its own service

Func/onali/es
From Waterfall to DevOps

Features, performance improvements, bug fixes, etc., are **periodically** delivered as one **big update**.

**Continuous** delivery of **incremental updates**. User feedback is constantly monitored and incorporated.

**Deploys 100 a day!**

**Deploys 300 a day!**
Microservices + DevOps

• Application is now a composition of loosely coupled microservices

• Individual microservices are owned and operated by independent teams

• Services are updated frequently, independent of other services, while maintaining compatibility
The Availability Problem

- Frequent performance issues due to insufficient testing
- High MTTR
  - Triaging is time consuming as knowledge base is spread across teams
- End user experience is impacted until the issue is fixed
An Interim Measure

• Downgrade parts of app, i.e., one or more services to an older version until symptoms disappear

• Autonomic downgrade for immediate response to availability related events
Assumptions

• Performance monitor
  – Source of truth for app health
  – Measures end user experience using various site metrics

• Faults handled
  – Prolonged periods of poor response time
  – High error rates from service API calls
  – Frequent crashes of service

• Faults not handled
  – Infrastructure issues
App-Bisect: Autonomous Healing

- Evaluate all possible compatible & older versions of service combinations in production, alongside current deployment.
- Stop search upon finding the most recent combination of versions that offer the desired end user experience.

Revision histories
Compatible Combinations of Microservices

- Leverage version dependencies across services
- When reverting a service to its earlier incarnation, revert dependent services to their most recent and compatible version
- Ensures that a deployed combination of services is compatible

![Diagram showing compatible combinations of microservices](image)

App Manifest file

```json
{
  "name": "A",
  "version": "5.0",
  ...
  "dependencies": {
    "B": ">= 3.0",
    "D": ">= 1.0 || <= 2.0"
  }
  ...
}
```

Dependency range
Bounding Search w/ Global Restore Points

- Global restore points
  - Created by developers
  - Signify points in application’s history where major updates were made, e.g., schema changes
  - Cannot revert beyond this point
- Search starts from current version to the most recent global restore point
Evaluating Downgraded Versions

- Version-aware routing using SDN infrastructure
- Deploy & route portion of user traffic through an older service chain
- Monitor application metrics for error symptoms
Routing Across the Right Versions

• Leverage the SDN substrate

• Uniquely identify services of a given version using <Machine-IP, SDN-switch-port>

• Setup routes according to the combination of versions being tested

• Not the most cleanest solution, but requires no modifications to the application
Open Issues

- Application abstractions for version-aware aware routing
- Dealing with state (data stores) when testing older versions of services
Thank You