DON’T CREATE A SERVICE MESS,
USE A SERVICE MESH

Exploring microservice challenges and solutioning tips

Jason Dudash
Solutions Architect & Open Source Builder
May 2019
Adopting microservices means accepting architectural complexity in order to gain agility.
DISTRIBUTED COMPUTING IS COMPLEX

...but it’s just groups of networked computers right?

The fallacies of distributed computing

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.

https://observablehq.com/@duncangeere/structure-de-quadrilateres
COMPLEXITY = PROBLEMS
SO IF YOU’RE BUILDING SOMETHING LIKE THIS

https://www.youtube.com/watch?v=CZ3wluvmHeM
YOU HAVE SIMILAR CHALLENGES
PARTIAL OUTAGES & CASCADING FAILURES

- Functionality can require a chain of services
- Any of these services can fail individually
  - permanently or temporarily
- Calling code should handle failures
  - Failures aren’t instantaneous
- Failures can cascade through services

Typically result in hanging and unresponsive app user interfaces
TRAFFIC SPIKES

- Traffic spikes are unpredictable increased load
- User demand or malicious attacks
- These spikes can be hard on specific services
- API gateways regulate at ingress only

Typically result in service failures

https://www.youtube.com/watch?v=hNjdBSola8k
COMPLEX DEPLOYMENTS & FAILURES

- Deploying changes are often a failure point
- Untracked deployments is a problem
  - “Our process docs” shouldn’t be the solution
- Service versioning adds complexity
  - SvcX:v1 isn’t compatible with SvcY:v2
- Problems can show up only after release

Typically result in downtime or degraded functionality
MORE PROBLEMS

1. Overloaded services
2. Anemic domain models
3. Usage limiting for limited resources
4. Versioning and version specific traffic
5. Service discovery
6. Integration testing
7. Malicious requests
8. Dealing with stateful services
9. Finding data from 100s of logs files
10. Services coupling leading to a Death Star
11. Inability to measure/monitor performance
12. New client libs force rebuild/test of services
13. Network Latency
14. Migration from a monolith
15. Identifying unhealthy services
16. Heterogeneous communication protocols
17. Managing retry logic
18. Testing failure conditions
19. Validation of users across services
20. Sneaky bugs that only rear their head in prod
84. Microservice libs unavailable for my language
85. Increased complexity for operations team
86. Code pipelines for 100s of services
87. Finding root cause of a failure
88. Controlling access to service APIs
89. Increased complexity for developers
90. Deployment failures
91. Distributed transactions
92. Insecure communication of data in-transit
93. Legacy data sources
94. Service level security auditing
95. Multi-region deployments
96. Tracing service call chains
97. Lack of role based access control
98. Data consistency
99. Insecure communication of data in-transit

I GOT 99 PROBLEMS, BUT A MESH AIN’T ONE
ISTIO - A SERVICE MESH

Connect
Intelligently control the flow of traffic and API calls between services, conduct a range of tests, and upgrade gradually with red/black deployments.

Secure
Automatically secure your services through managed authentication, authorization, and encryption of communication between services.

Control
Apply policies and ensure that they're enforced, and that resources are fairly distributed among consumers.

Observe
See what's happening with rich automatic tracing, monitoring, and logging of all your services.
The **data plane** is composed of a set of intelligent proxies (Envoy) deployed as sidecars that mediate and control all network communication between microservices.
The **control plane** is responsible for managing and configuring proxies to route traffic, as well as enforcing policies at runtime.
THIS IS THE ISTIO SERVICE MESH

A network of deployed services with load balancing, service to service authentication, monitoring, (and more), requiring few or no code changes to your services source code.

This is possible due to data plane sidecars and control plane components.
ADDRESSING COMMON PROBLEMS BY...
CIRCUIT BREAKING AND BULKHEADS

- Fail FAST - prevent requests to failed services
- Configure via Istio’s “DestinationRule“
- outlierDetection
  - Check for consecutive errors
  - Applied to HTTP and TCP traffic
- connectionPool
  - Can use HTTP or TCP
- You get immediate 503s when tripped
RATE LIMITING

- Stop traffic from overloading services
- Define quotas to limit routing
  - Tune for individual services
  - e.g. Requests per second
- Apply rules for more control
  - e.g. Quota only apply anon users
- Add autoscaling to help meet demand

```yaml
quotas:
  - name: requestcountquota.instance.istio-system
    maxAmount: 500
    validDuration: 1s
    overrides:
    - dimensions:
      | destination: serviceA
      | maxAmount: 1
      | validDuration: 5s
    - dimensions:
      | destination: serviceB
      | source: "10.28.11.20"
      | maxAmount: 500
      | validDuration: 1s
    - dimensions:
      | destination: serviceB
      maxAmount: 2
      validDuration: 5s
```
MIRRORING / TRAFFIC SHIFTING

- Mirror traffic to new and old for prod. env tests
  - Get real data into the new version to test
- Slowly shift traffic to new service versions
  - Minimal load reduces impact of issues
  - Shift back if needed
- Configure via Istio’s "VirtualService"
VERSION BASED ROUTING
AUTOSCALING
STAGED ROLLOUTS
CANARY DEPLOYMENTS
BLUE/GREEN DEPLOYMENTS
DISTRIBUTED TRACING (JAEGGER)
VISUAL SERVICE HEALTH (KIALI)
DISTRIBUTED LOGGING
COLLECTING AND VISUALIZING METRICS
AND MORE...
WHY IS THIS EXCITING?

- It’s a lot of capability
- It’s dynamic
- It’s automatic
- It’s configurable
- It’s not in the code
- It’s polyglot

And it’s bundled as an installable platform for OpenShift / Kubernetes!

AND IT’S BUNDLED AS AN INSTALLABLE PLATFORM FOR
OPENSHIFT / KUBERNETES!
TODAY I LEARNED...
TODAY I LEARNED

- Adopting microservices comes with new challenges
- Don’t force those challenges onto developers
- Follow best practices and leverage
  - Circuit breaking and bulkheads
  - Advanced traffic management
  - Distributed tracing, logging, and metrics
  - Load balancing and autoscaling

Istio can take operational burden off a team by addressing common challenges with little to no code change
FIND OUT MORE AND TRY IT YOURSELF

https://learn.openshift.com/servicemesh

https://developers.redhat.com/topics/service-mesh/
https://developers.redhat.com/topics/microservices/

https://github.com/dudash/openshift-microservices