Signadot

Scale Environments in Kubernetes with OpenTelemetry + Service Mesh

Scale 21x
16 March 2024

Anirudh Ramanathan
CTO, Signadot

/ whoami

- Co-founder & CTO of **Signadot**
  - Founded in late 2019
  - Building testing platform for microservices
  - Backed by Y-Combinator & Redpoint Ventures

- Previously worked on Kubernetes at Google
  - Worked on core controllers (StatefulSet, Deployment, etc)
  - Worked on primitives for batch & data processing in k8s
  - Committer on the Apache Spark project

/ agenda

- Introduction to Test Environments using OTel & Istio
- Concepts: baseline env, message queues, databases
- Implementation
- Practical considerations: how to roll this out & measure success in your organization

Introduction

/ challenges with testing microservices

- **Complex Interdependencies**
  - Talking to each other via APIs and with 3rd party dependencies
  - May be independently released
- **Fast Feedback**
  - Getting high quality feedback early in the development lifecycle
- **Environment Consistency**
  - Is the environment close to production? Easy to maintain?
  - Does it have realistic data & 3rd party interactions?
  - Observability & debugging

/ testing: fast feedback loops

fast & cheap ---------- slow & expensive

SDLC Phase:	local workstation	CI	Staging	Production
Persona(s):	Dev	Dev	Dev, QA, PM	Customers/Partners

For testing to be effective, we need clear signal as early as possible.

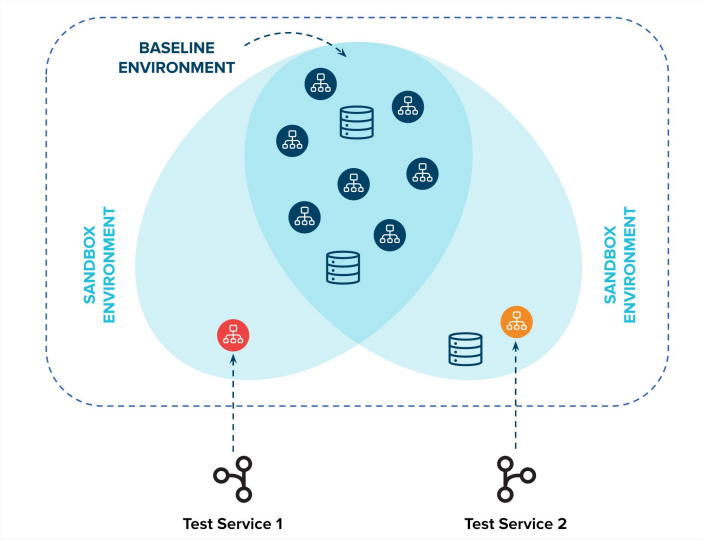
/ testing: scaling environments is hard

Multiple pre-prod environments

- Easy to roll out with small footprint but hard to scale & maintain
- Infra costs become prohibitive (#microservices x #environments)
- Operational burden (e.g. observability, lifecycle management) grows rapidly
- Drift between different environments reduces confidence in feedback
- Time to set up increases, impacts dev productivity
- Not realistic data

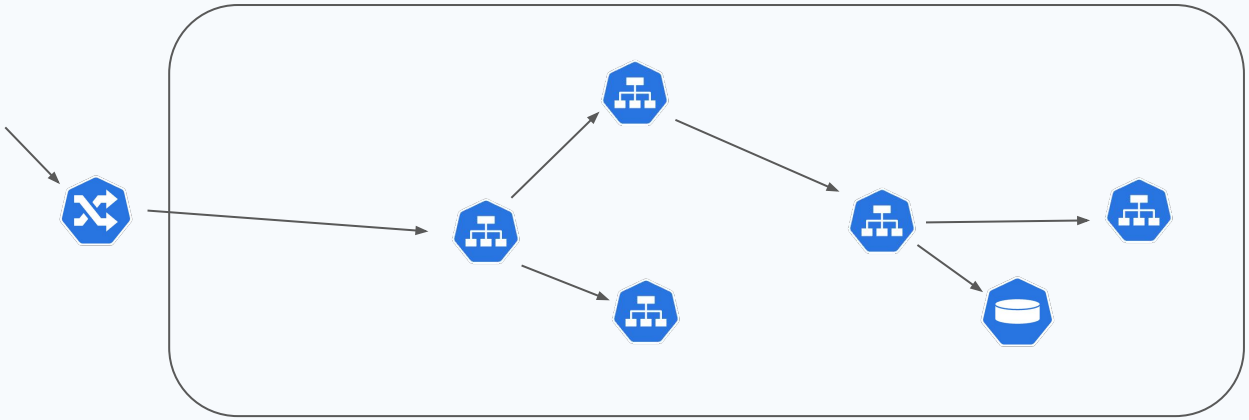
/ testing with sandboxes

- Combines the test (or sandboxed) versions of microservices with **shared pool of dependencies.**
- Dependencies updated with the stable versions of code by a CD process.
- Each test **sandbox** requires only deploying “what changed”, unchanged dependencies satisfied from shared pool.



Concepts

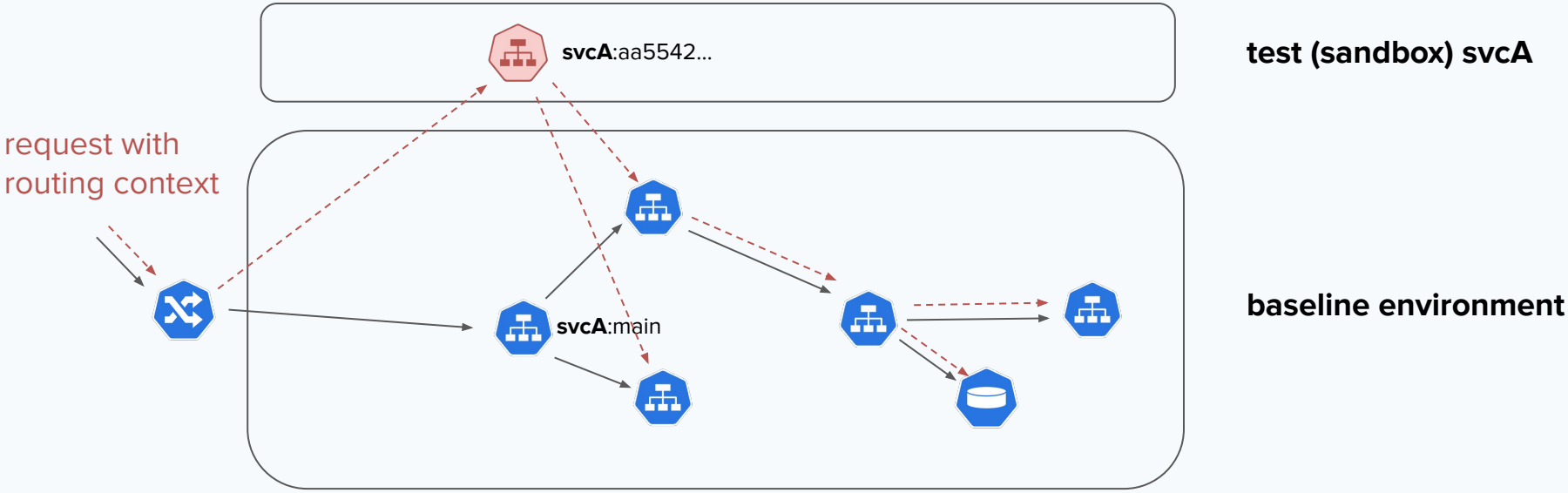
/ request isolation: testing using sandboxes



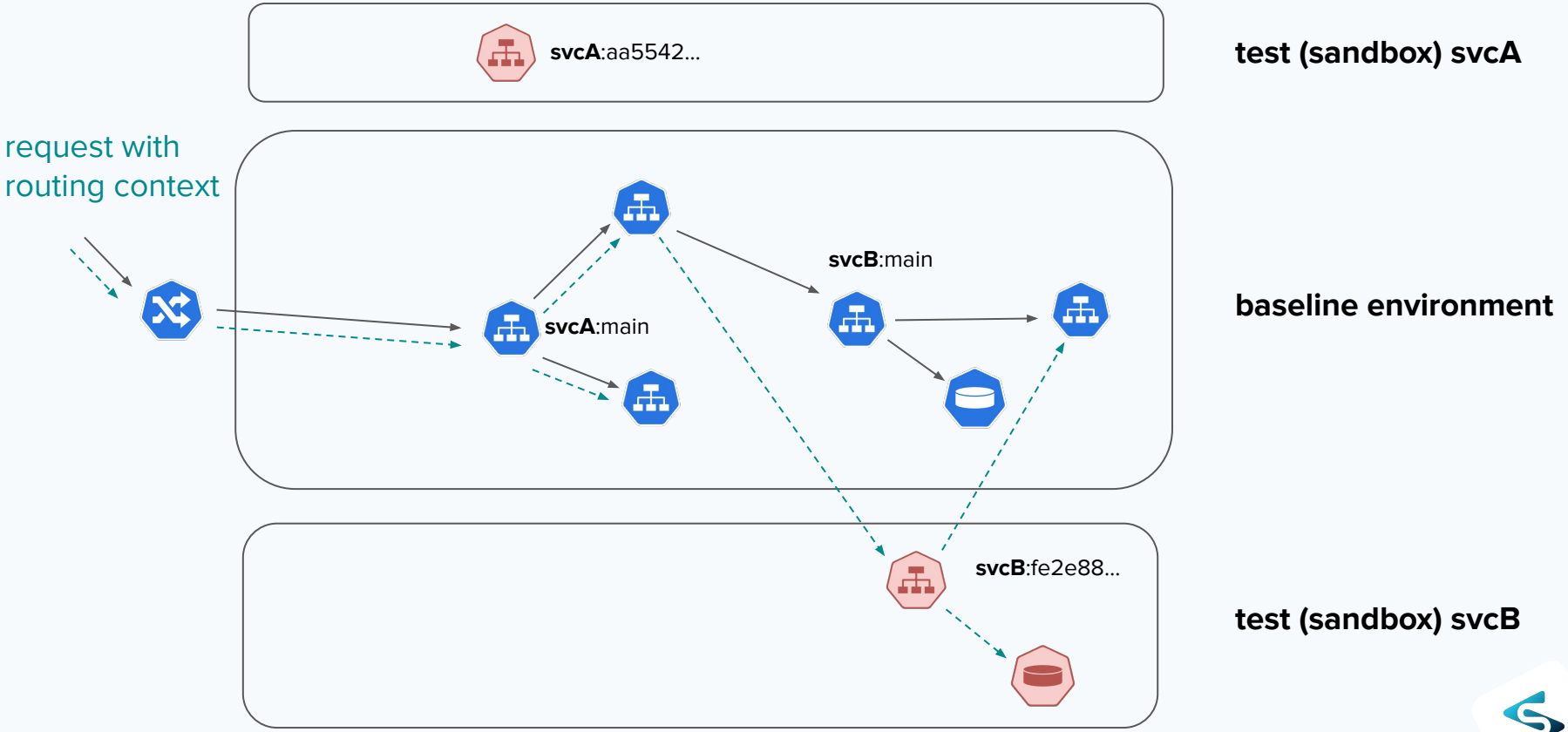
baseline environment

every microservice continuously updated by CD process to stable versions

/ request isolation: testing using sandboxes



/ request isolation: testing using sandboxes

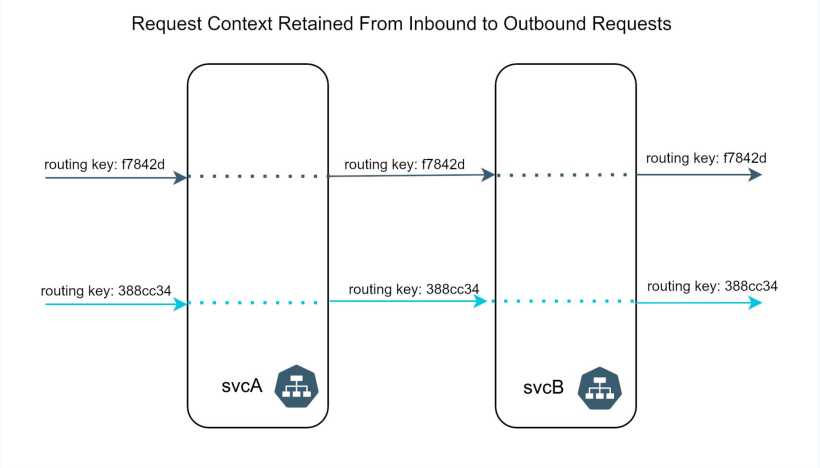


/ context propagation using OpenTelemetry

- No tracing backend needed. OTel Supports https://www.w3.org/TR/baggage/ out of the box

- Java / Javascript / Python / PHP / .NET support auto-instrumentation without code changes

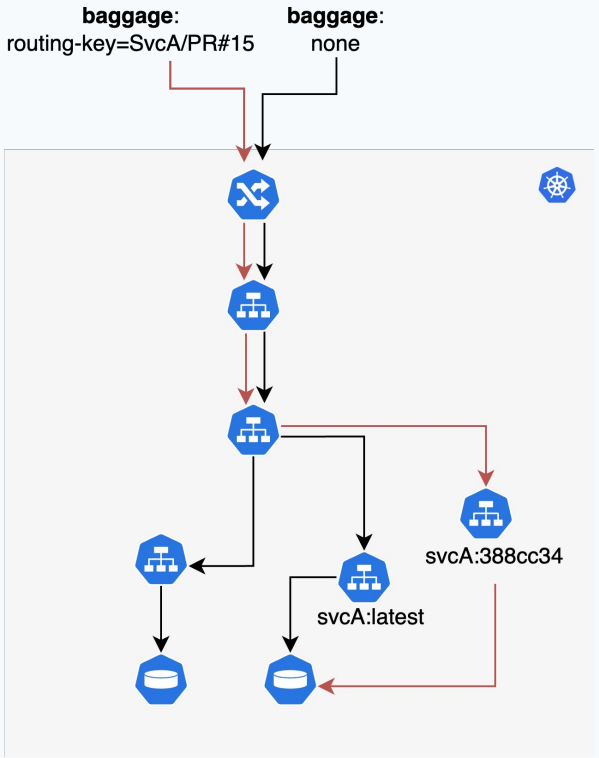
- Other languages may need code changes
  - **Pattern**: Platform Teams create middleware / libraries that make this easier for product teams to adopt



https://opentelemetry.io/docs/languages/

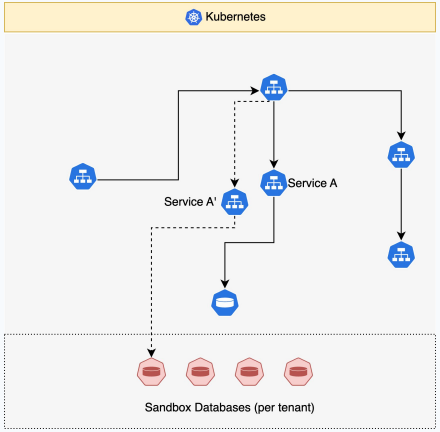
/ request routing using service mesh

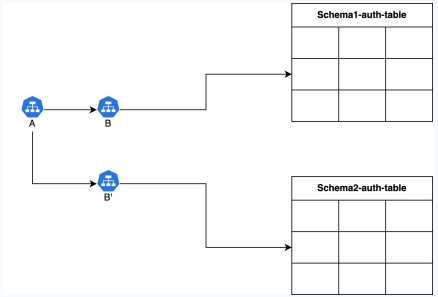
- Configure Routing using Service Mesh (e.g. Istio) layer and the baggage header
- Test from any service once context propagation in place, even the frontend using existing URL + Header
- L7 protocols: HTTP, gRPC typically supported out of the box



/ sandboxing databases

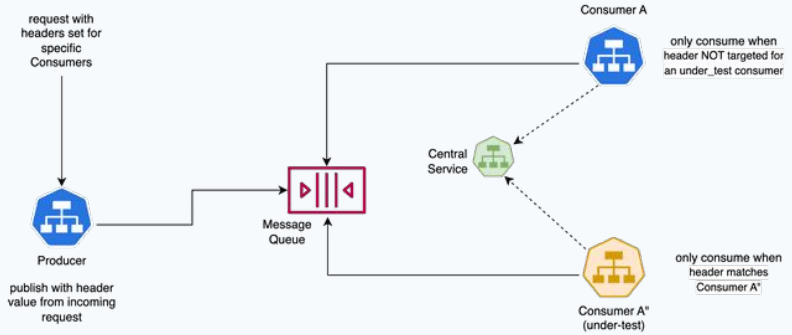
- For E2E testing and Exploratory Testing (Previews)
  - Users use high-quality data in staging
  - Isolation in entity domain (test orgs, test users, etc)
- If additional isolation needed for automated tests / schema changes
  - Not always infra level isolation: can be at the logical level (table, database, etc)
  - Ephemeral databases with staging snapshots attached at runtime to sandboxed workloads via env vars
  - **Advanced**: Put tenancy info into the database tables themselves at the row level





/ sandboxing message queue flows

- Teach consumers to consume messages with awareness of tenancy
- Producer injects routing context into message metadata via OTel
- Consumers are made aware of tenancy using env vars
  - Different consumer group for baseline and sandboxed consumers
  - Sandbox consumers reject baseline messages & vice versa

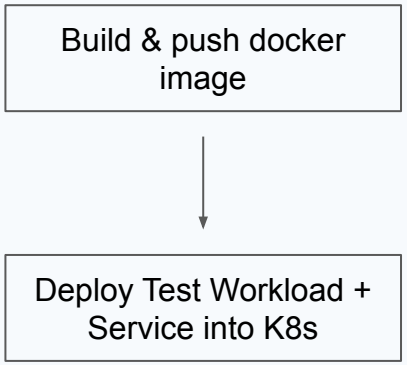


https://thenewstack.io/testing-event-driven-architectures-with-opentelemetry/

Implementation

/ creating the test workload

- Build a new docker image for version of microservice you want to test
- After image is built, create a new Kubernetes Workload (Deployment, Argo Rollout, etc)



This whole process is often tied to CI/CD so that this happens automatically when a Pull Request is opened

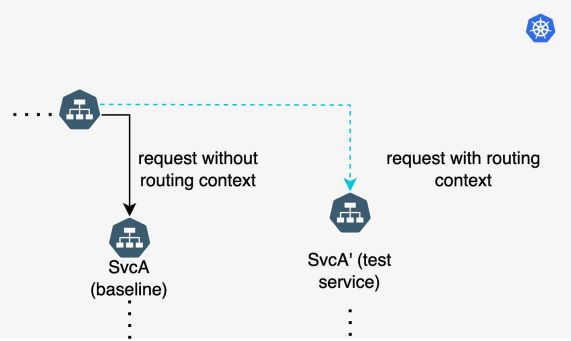
/ specifying the sandboxed workload

- It's possible to just specify "what changed" with respect to the baseline workload
- Deploy test version into the same namespace: reuse the same secrets and configmaps used by the baseline workload
- Corresponding to the workload, we will also need a K8s service corresponding to it for setting up request routing

```
name: '@{name}'
spec:
  labels:
    team: backend
  description: sandbox env for mysvc
  cluster: staging-1
  forks:
  - forkOf:
      kind: Deployment
      name: mysvc
      namespace: default
    customizations:
      images:
      - image: repo/image:@{gitsha}
      env:
        debug: true
```

/ request routing using istio

- Configure Istio VirtualServices to route based on baggage header value
- **Pattern**: prevent CD system from overwriting these changes

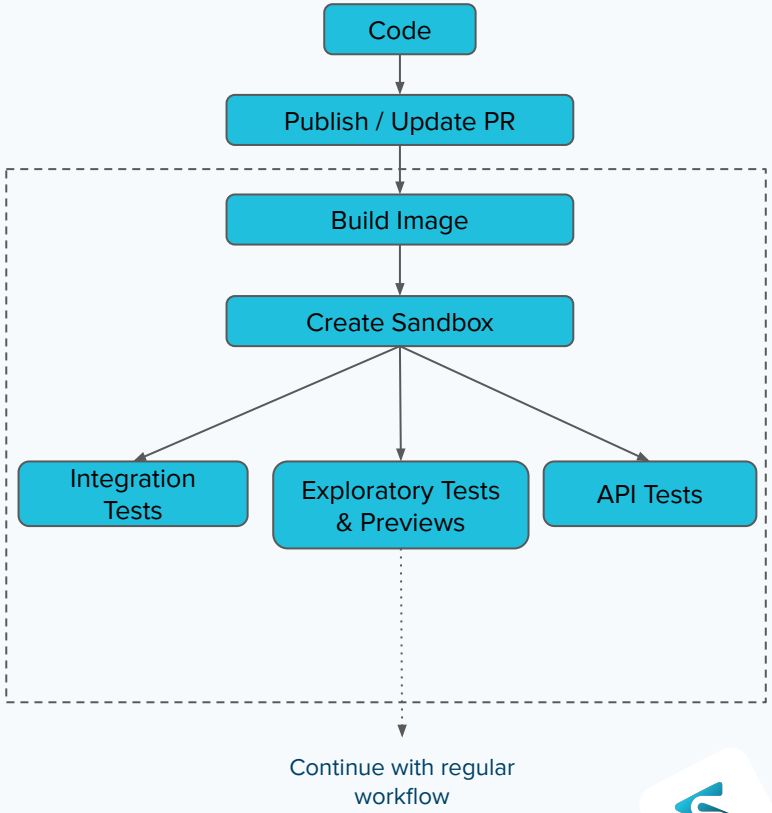



```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata: ...
spec:
  hosts:
  - route
  http:
  - match:
    - headers:
        baggage:
          regex: ^.*\b(sd-routing-key|sd-sandbox)\s*=\s*rfd5sxlnwldzu\b.*$
      port: 8083
    name: signadot-operator-ir-route-hotrod-istio-rfd5sxlnwldzu-8083
    route:
    - destination:
        host: pr-220-route-dep-route-c33c9d43.hotrod-istio.svc
        port:
          number: 8083
  - route:
    - destination:
        host: route
```

Practical Considerations

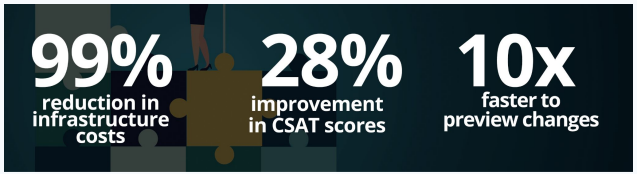
/ best practices for adoption

- Start with rolling out sandboxes associated with pull requests for a few backend microservices:
  - **Preview Environments** (use Chrome Extension to set header)
  - **E2E automation** testing critical functionality
- OTel coverage often has holes - so, expect gradual rollout
- Implement tenancy-based data isolation in microservices gradually

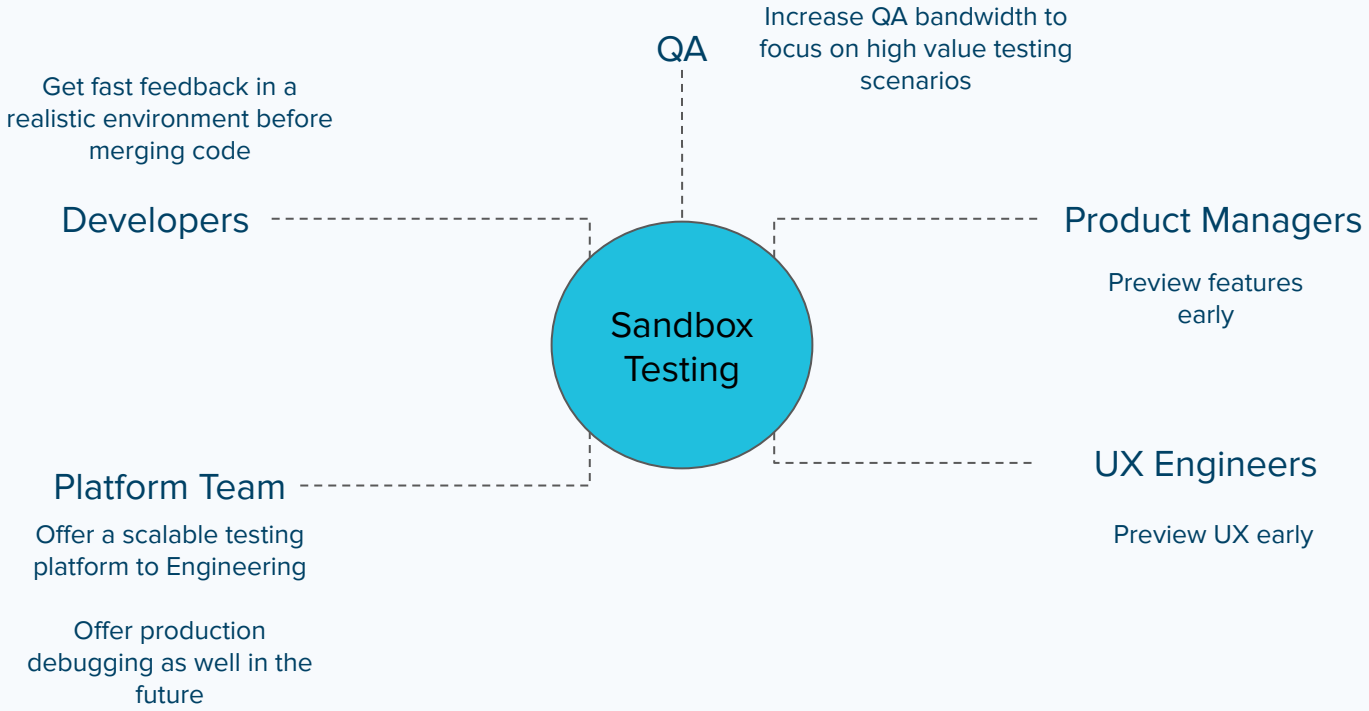


/ measuring success

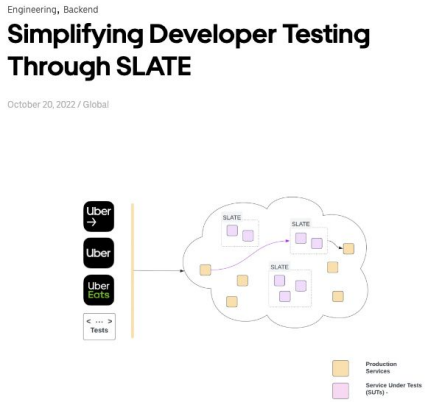
- DORA metrics can help
- Get qualitative feedback also using developer surveys & feedback sessions
- Measure # of microservice integration issues discovered on higher environments (staging / prod)



/ why use sandboxes for testing?

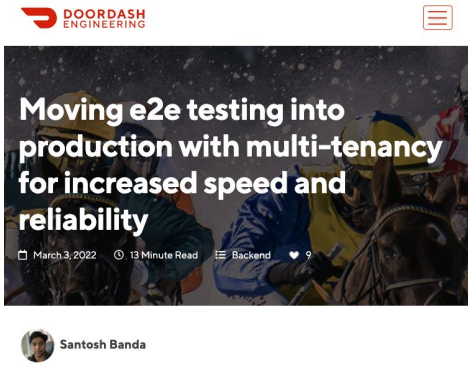


/ similar solutions operationalized at scale



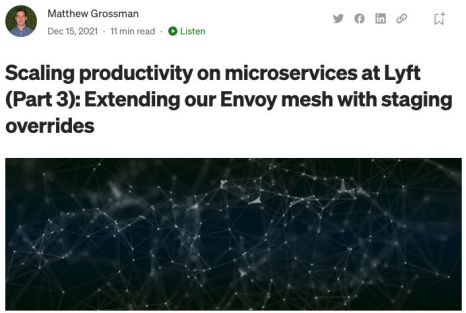
The performance of **Uber's** services relies on our ability to quickly and stably launch new features on our platform...

https://www.uber.com/blog/simplifying-developer-testing-through-slate/



When **DoorDash** moved from monolith to microservices, we needed a more scalable approach to production testing...

https://doordash.engineering/2022/03/03/moving-e2e-testing-into-production-with-multi-tenancy-for-increased-speed-and-reliability/



...how we scaled our development practices at **Lyft** in the face of an ever-increasing number of developers and services....

https://eng.lyft.com/scaling-productivity-on-microservices-at-lyft-part-3-extending-our-envoy-mesh-with-staging-fdaafafca82f

/ closing thoughts

- **“Local” Sandboxes**
  - Same methodology, but workloads live on developer workstations
- **Testing Features spanning across microservices**
  - Combine sandboxes together to test features (e.g. Backend + Frontend changes)
  - Faster feedback than using feature flags.
- **Testing in production**
  - Once there’s strong controls over data, you can enable testing with sandboxes in prod!
  - Sandboxes for developer testing in production being used at leading companies like Uber & Doordash!
  - Can be used for prod debugging as well

Quick Demo

Q&A

Anirudh Ramanathan (anirudh@signadot.com)

@foxish (GitHub)

@foxish_ (twitter)

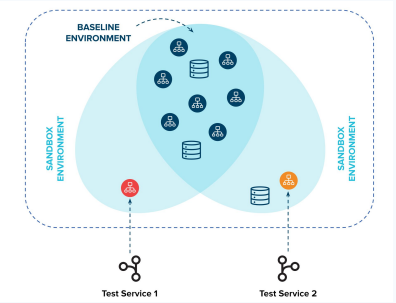
Signadot

www.signadot.com

/ request tenancy v/s other approaches

Multiple pre-prod environments

- Easy to roll out with small footprint but hard to scale
- Drift between different environments reduces confidence in feedback
- Less realistic data
- Infra costs & operational burden may become prohibitive (#microservices x #environments)



Sandbox Environments

- More gradual rollout may be needed
- Very small additional cost & infra burden for each environment
- Very fast to spin up & developer-friendly
- Covers many forms of testing & stabilizes shared environment