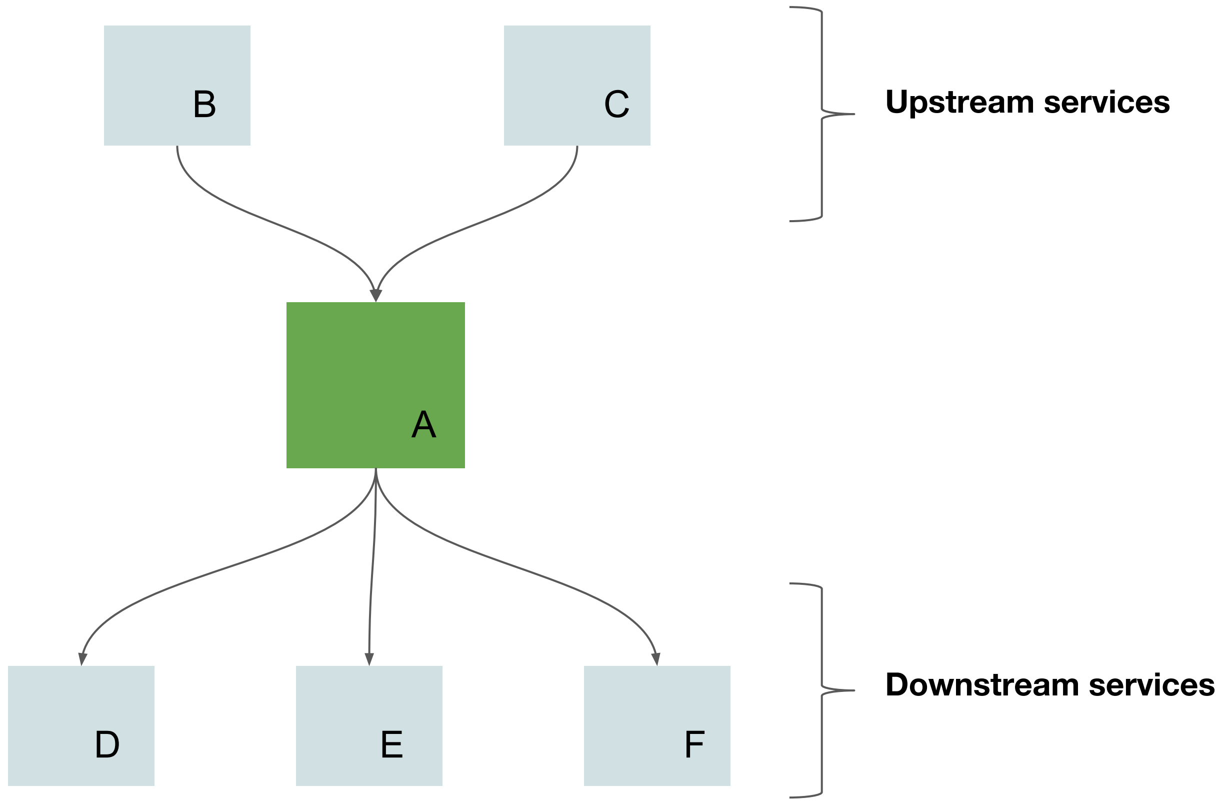
Testing in Production at Scale

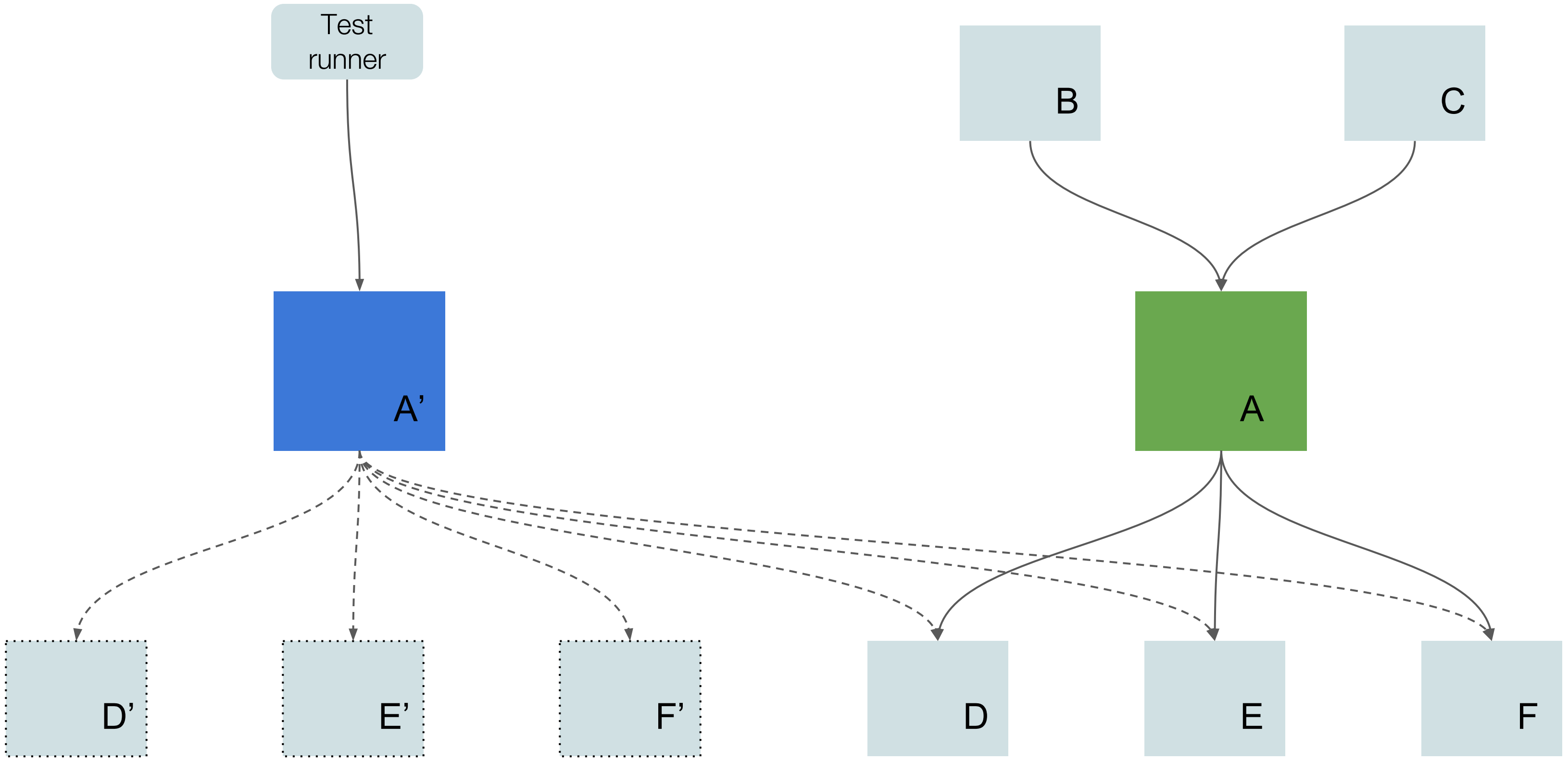
Uber

Amit Gud | SREcon19 Americas | March 25, 2019

Meet Alice!

Alice
Software Developer




Test runner
A'
D'
E'
F'
B
C
A
D
E
F

Key Takeaway

Testing in Production can be a viable solution.

Agenda

The Scale

600
Cities

64
Countries

75m
Active Riders

3m
Active Drivers

15m
Trips Per Day

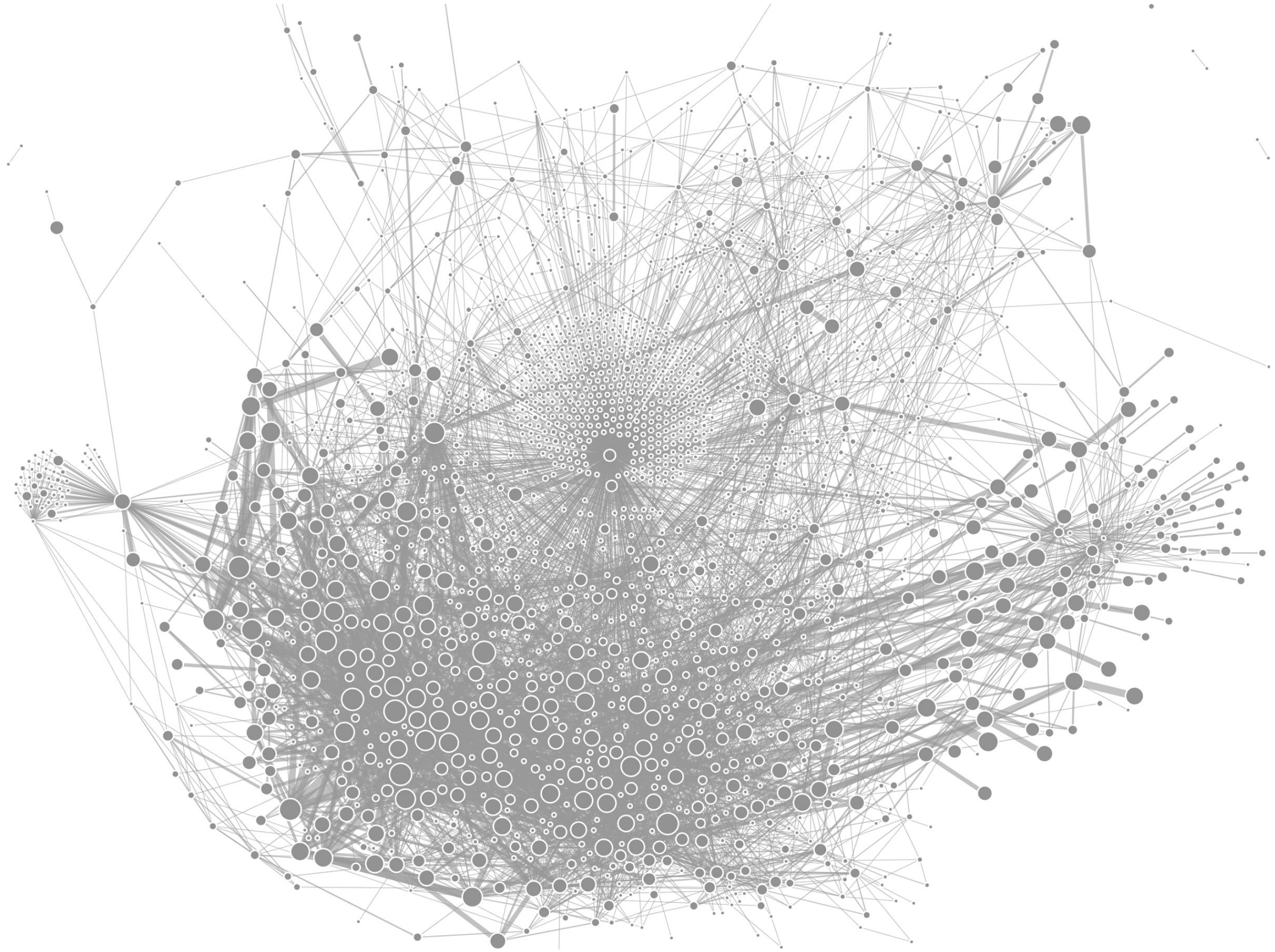
10b
Cumulative Trips

1000s

Microservices

1000s

Commits per day



Agenda

Why Test in Production?

Less operational cost of maintaining a parallel stack.

One knob to control capacity.
No synchronization required.

Why Test in Production?

More accurate end-to-end capacity planning.

Delta test traffic runs on the production stack.
Test traffic takes same code path as production traffic.

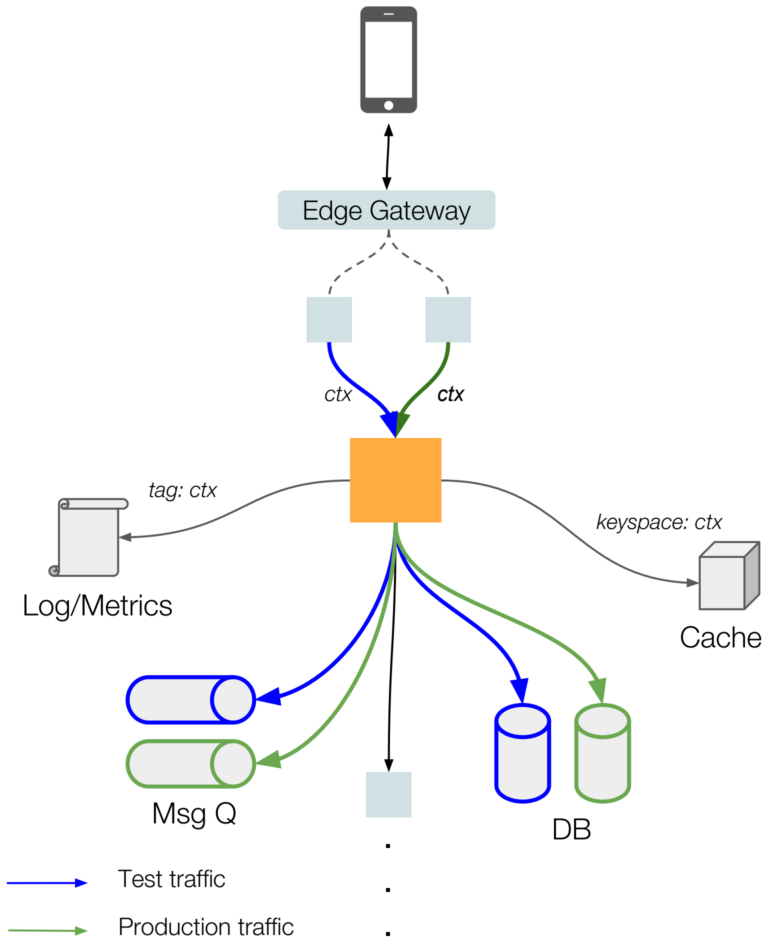
Bonus: The Testing in Production framework enables other use case.

Use cases like Canary, Shadowing, A/B Testing become an extension to the Testing in Production framework.

Agenda

Tenancy Oriented Architecture

- Isolation between test & production
- Tenancy-based access control
  - Test request cannot create/mutate prod artifacts
- Minimal deviation between test and production environments



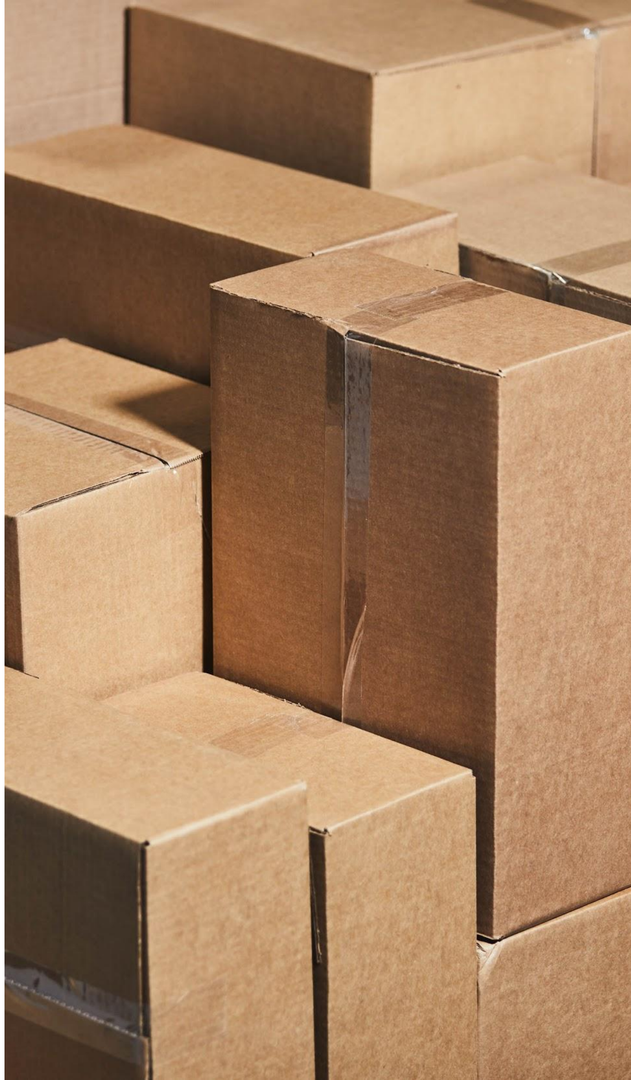
Design Considerations

- Infra components needing tenancy support
- Explosion of support matrix
    - # of transports/encodings
    - # of languages
- Gradual transition from current architecture to tenancy-aware architecture
- Tenancy-based service discovery & routing
- Onboarding overhead - impact on developer productivity

Agenda

Tenancy Building Blocks

1. Context & Context propagation
2. Tenancy Aware Infrastructure
3. Tenancy Aware Environments
4. Tenancy Aware Routing



1. Context & Context Propagation

- Tenancy context for both in-flight data (requests) and the at-rest data (persistent artifacts)

- Tenancy can be ‘testing’, ‘production’, etc.
    - Aligns with tenancy of the actors involved in the request

- Request tenancy propagated agnostic to transport / protocol

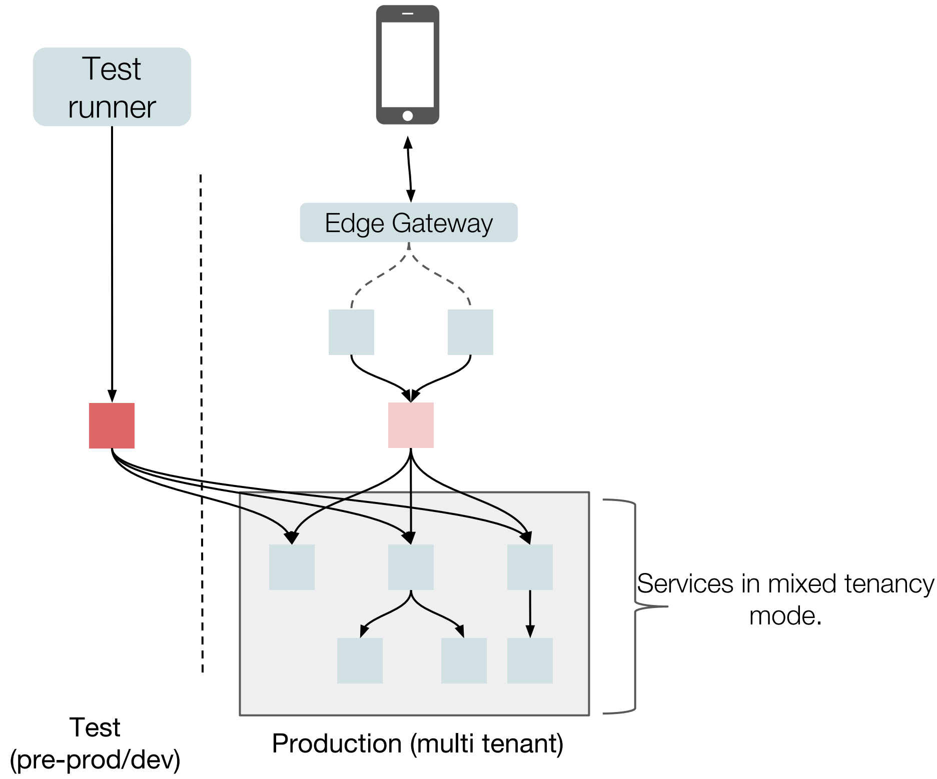
- Persistent artifact tenancy implementation depends on the specific data component

2. Tenancy Aware Infrastructure

- Types of infrastructure components
  - Storage datastores, e.g. Cassandra
  - Message queues, e.g. Kafka
  - External caching, e.g. Redis
  - Search, e.g. ElasticSearch
  - Observability: Logging, Metrics.
- 2 ways of making infrastructure aware of tenancy
  - Client library (language specific)
  - Gateway integration

3. Environments - Mixed Tenancy Mode (Goal State)

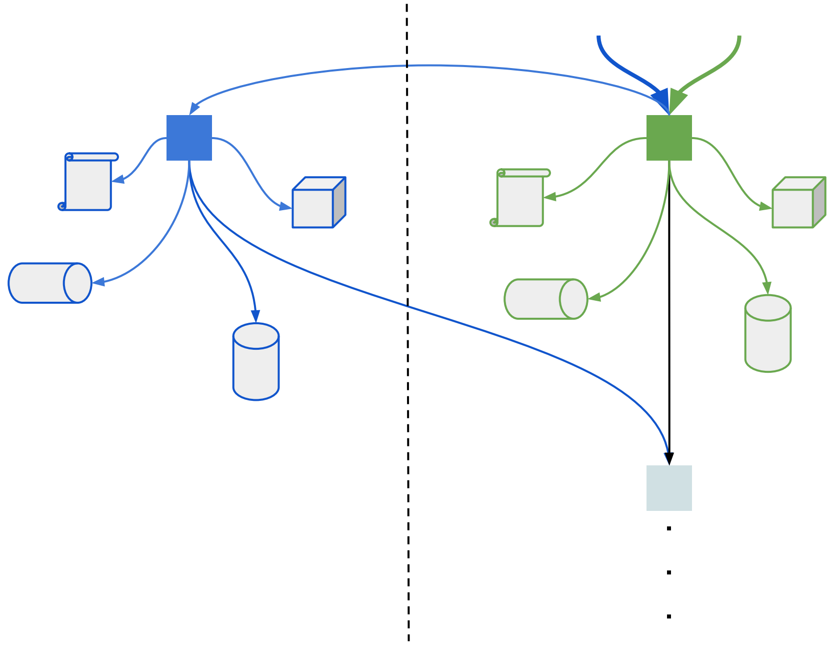
- Every service instance is able to handle both test and prod traffic.
- “Native tenancy” support for all the infra components.



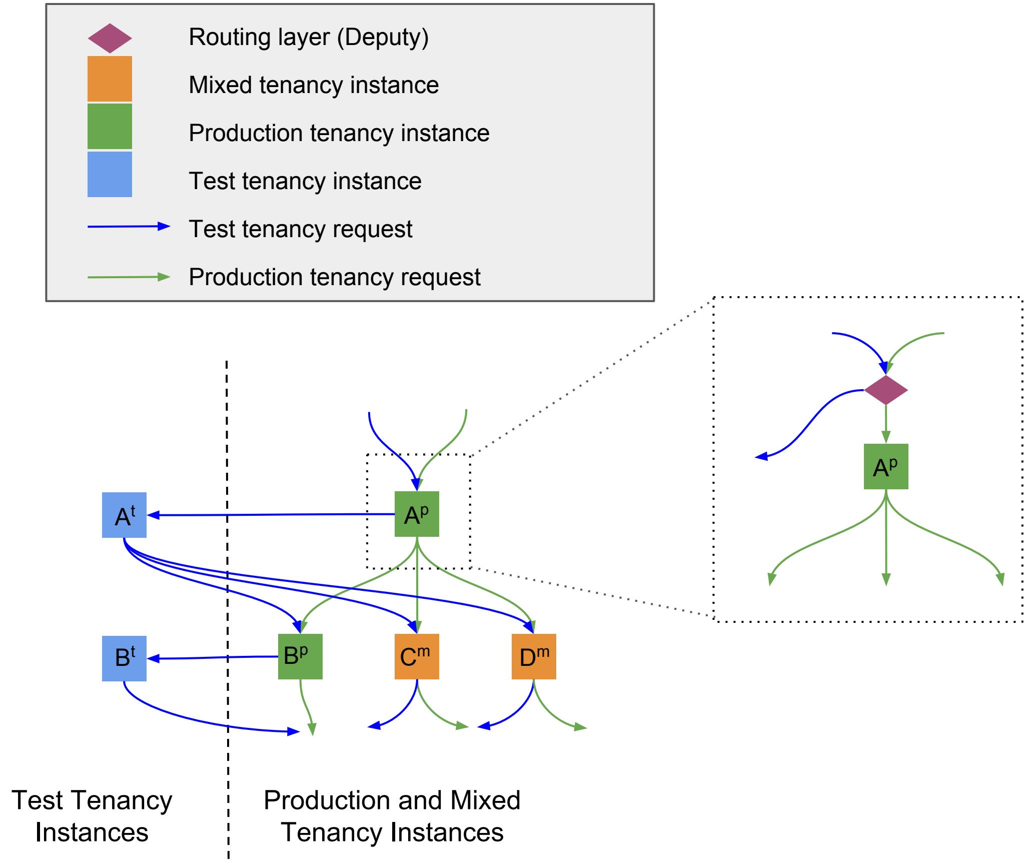
3. Environments - Test Tenancy Mode (Intermediate State)

- Supports tenancy adoption in advance of infra support.
- Separates the infra components explicitly via a separate environment.
- Utilize tenancy-based request routing to route test traffic to test tenancy environment.

🟩	Service instance in Production environment.
🟦	Service instance in Test Tenancy environment.
⬜	Downstream service.



4. Tenancy Aware Routing



- Out-of-process sidecar implementation.
- Agnostic to service language and transport used.
- Config-based routing policies and instant kill-switch.

Agenda

Extensions to Tenancy Architecture

- Rate Limiting
  - Tenancy-based QoS policies.
  - Safe-guard production from other traffic.
- Shadow traffic
  - Route traffic for A/B testing, where A is experimental code and B is production.
  - Ability to route only portion of the traffic without affecting production.
- Canary Deployments, Blue/Green Deployments
  - Gradually bring up/down deployments.
- Record & Replay
  - Duplicate part or whole of traffic to record requests for a particular scenario or user.

#TiP-is-not-as-scary-as-it-sounds!

Building a framework for Testing in Production is a long-term investment and can be a viable solution.

Thanks

Uber