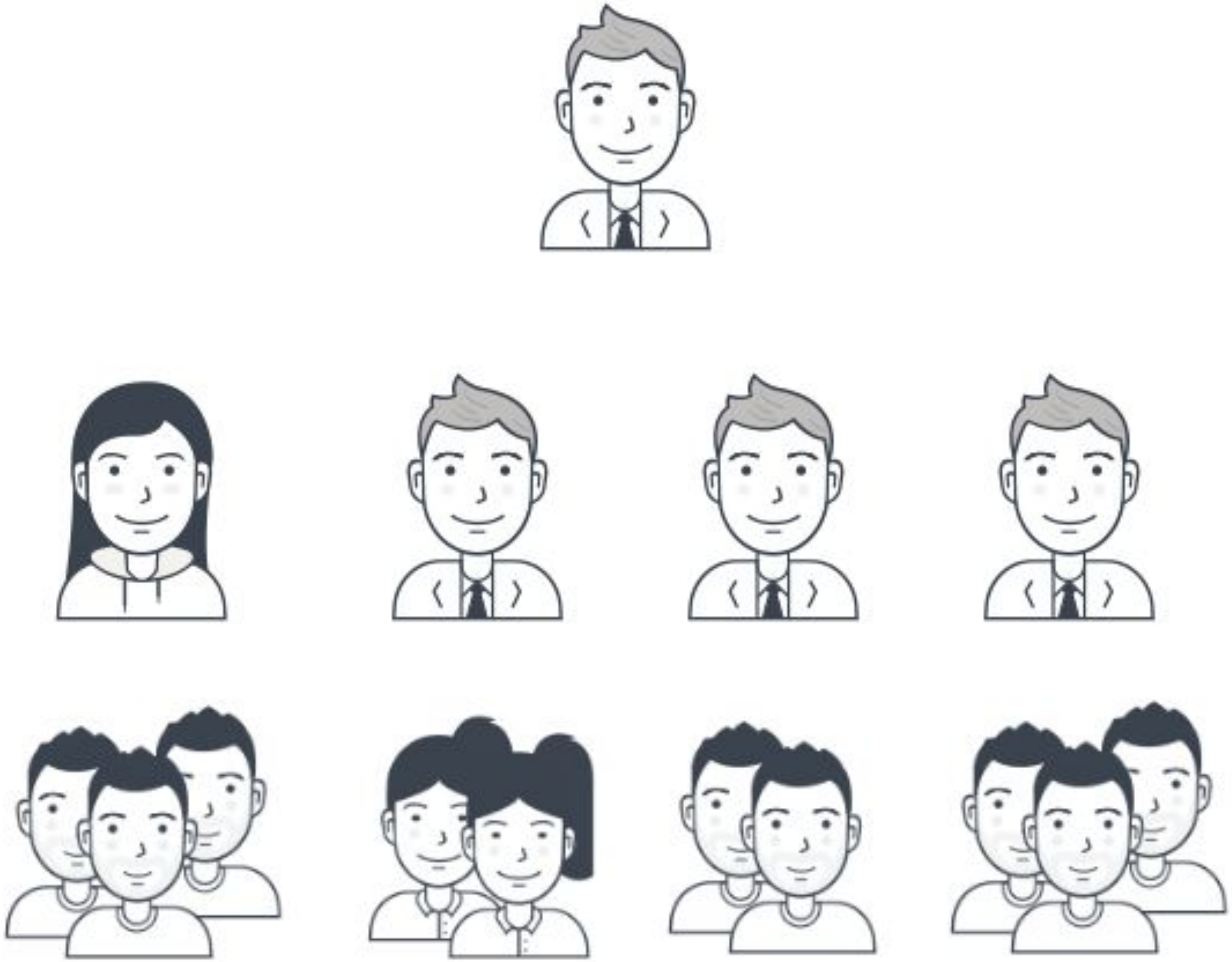
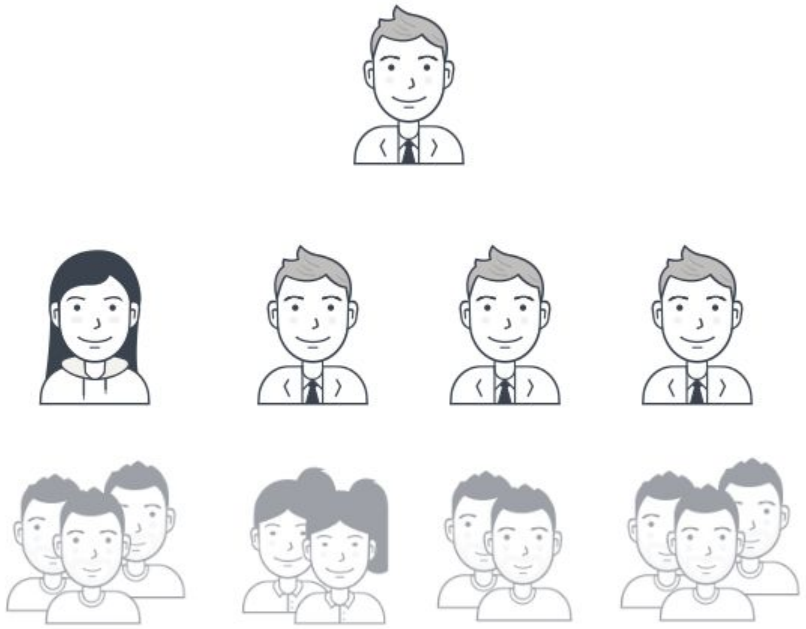
Honest, simple and fast isolation tests

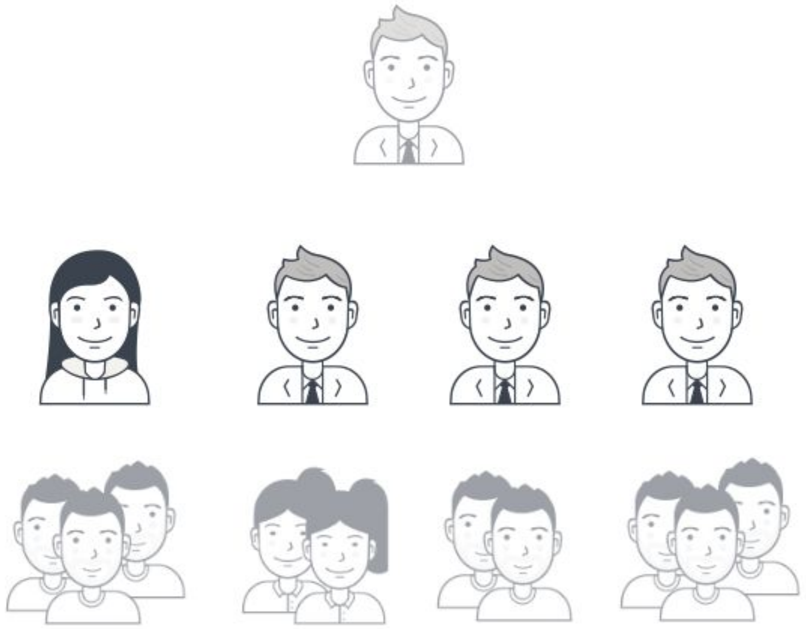
JUNO



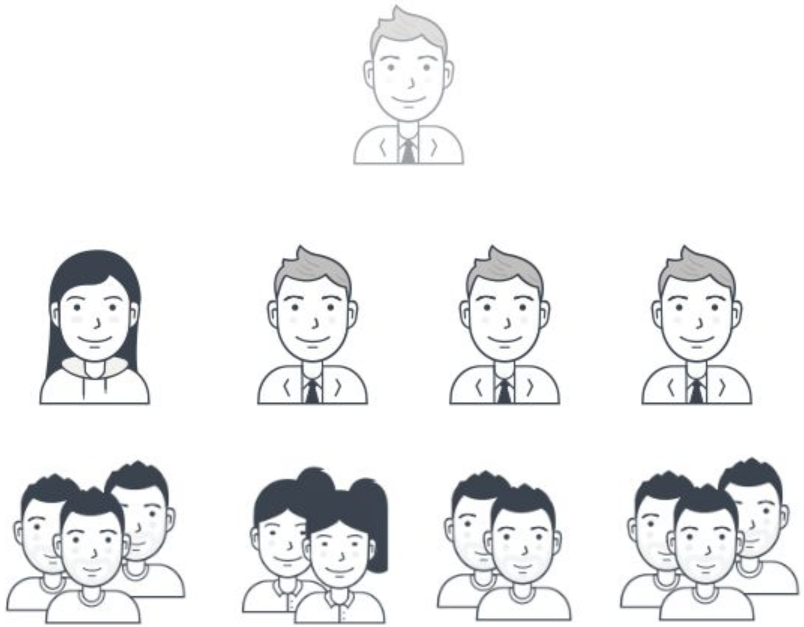
High level prioritization



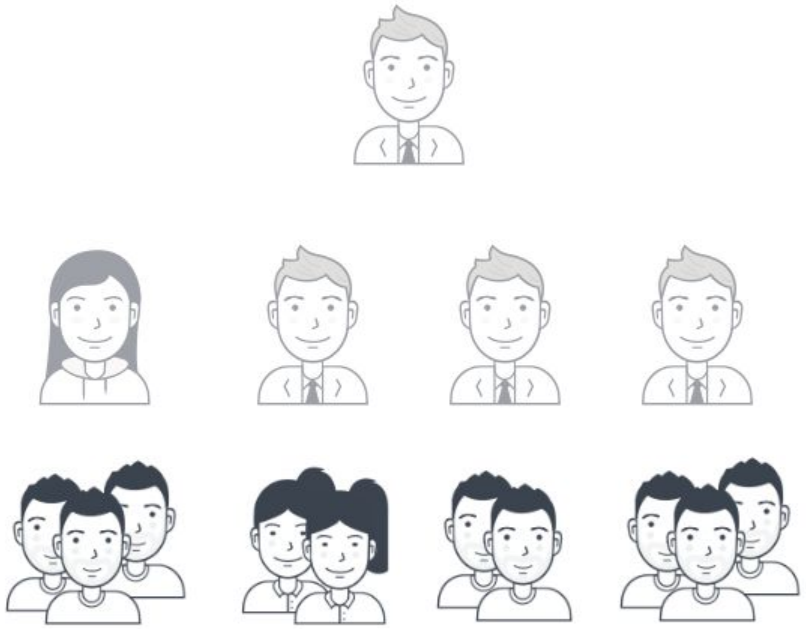
Task requirements



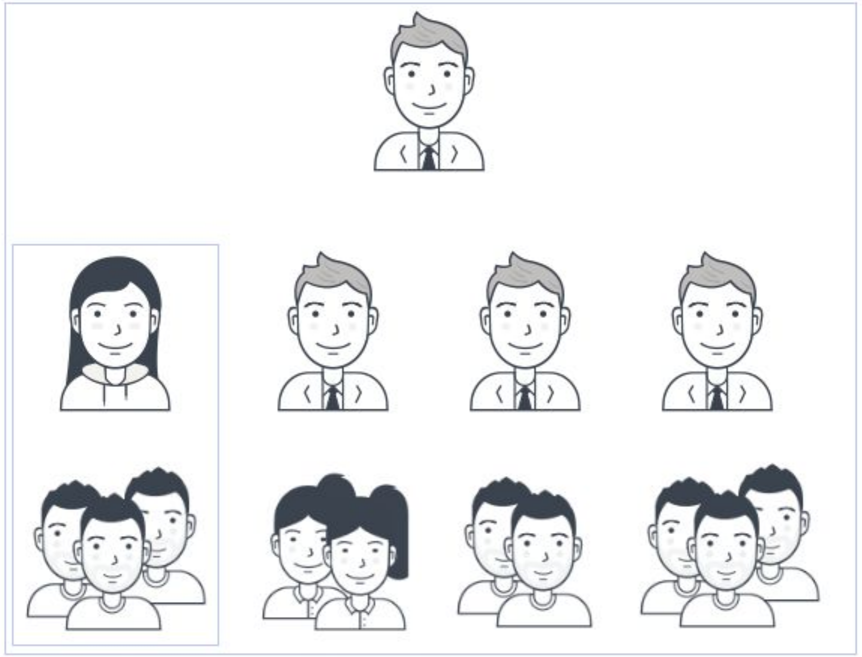
Dev code review



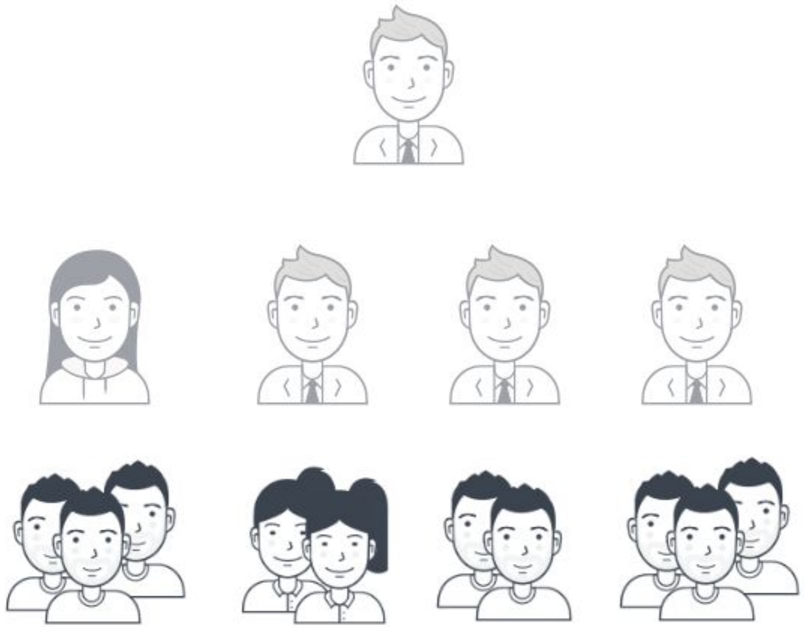
Fix/add tests for each code change



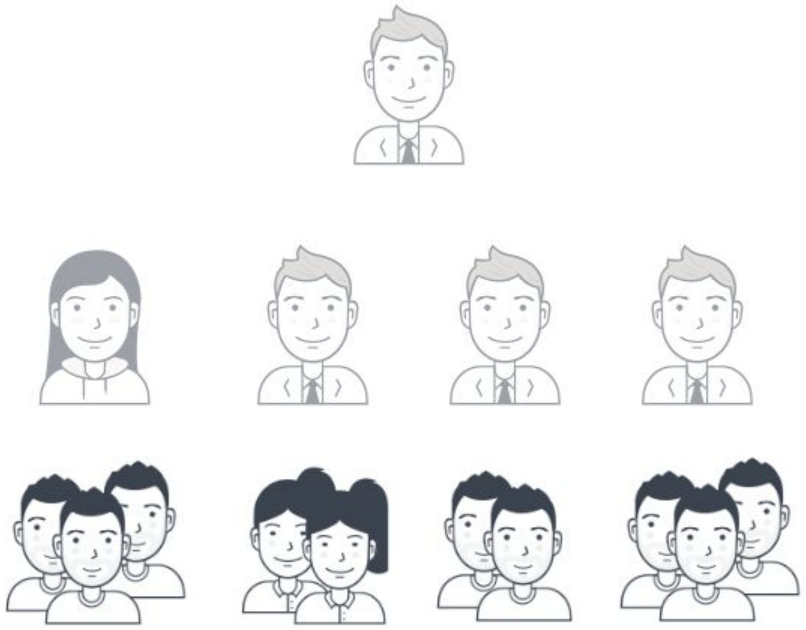
Review of test code



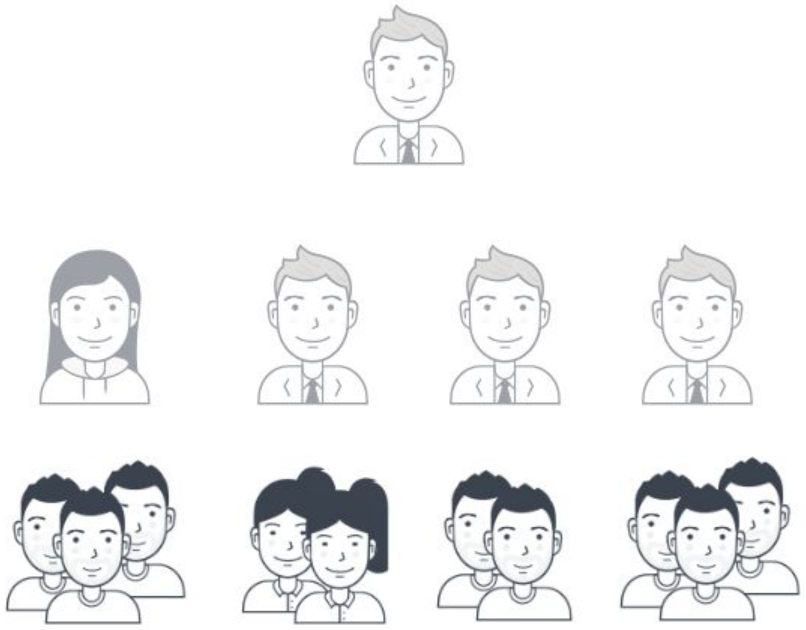
Run target and affected ms tests, merge test code



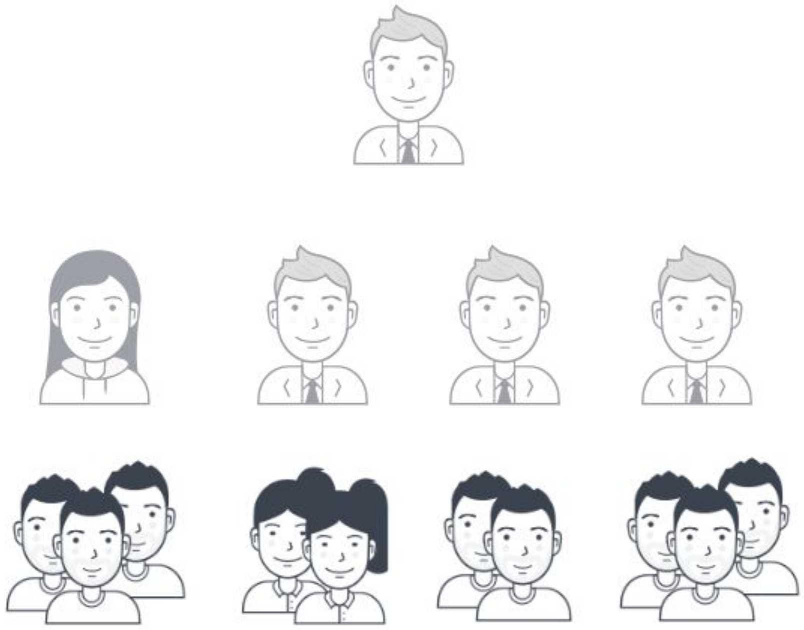
Check if all required microservices are deployed in staging



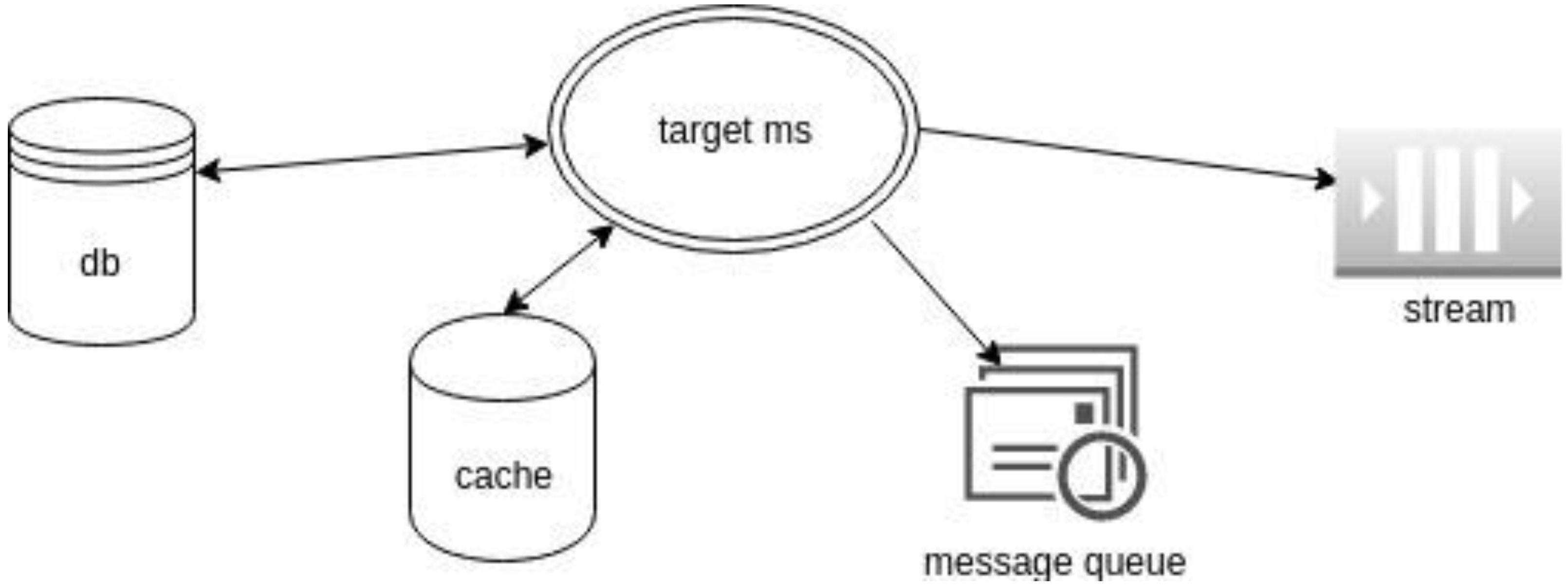
Assign to mobile/web QA if it is possible to check the task in clients



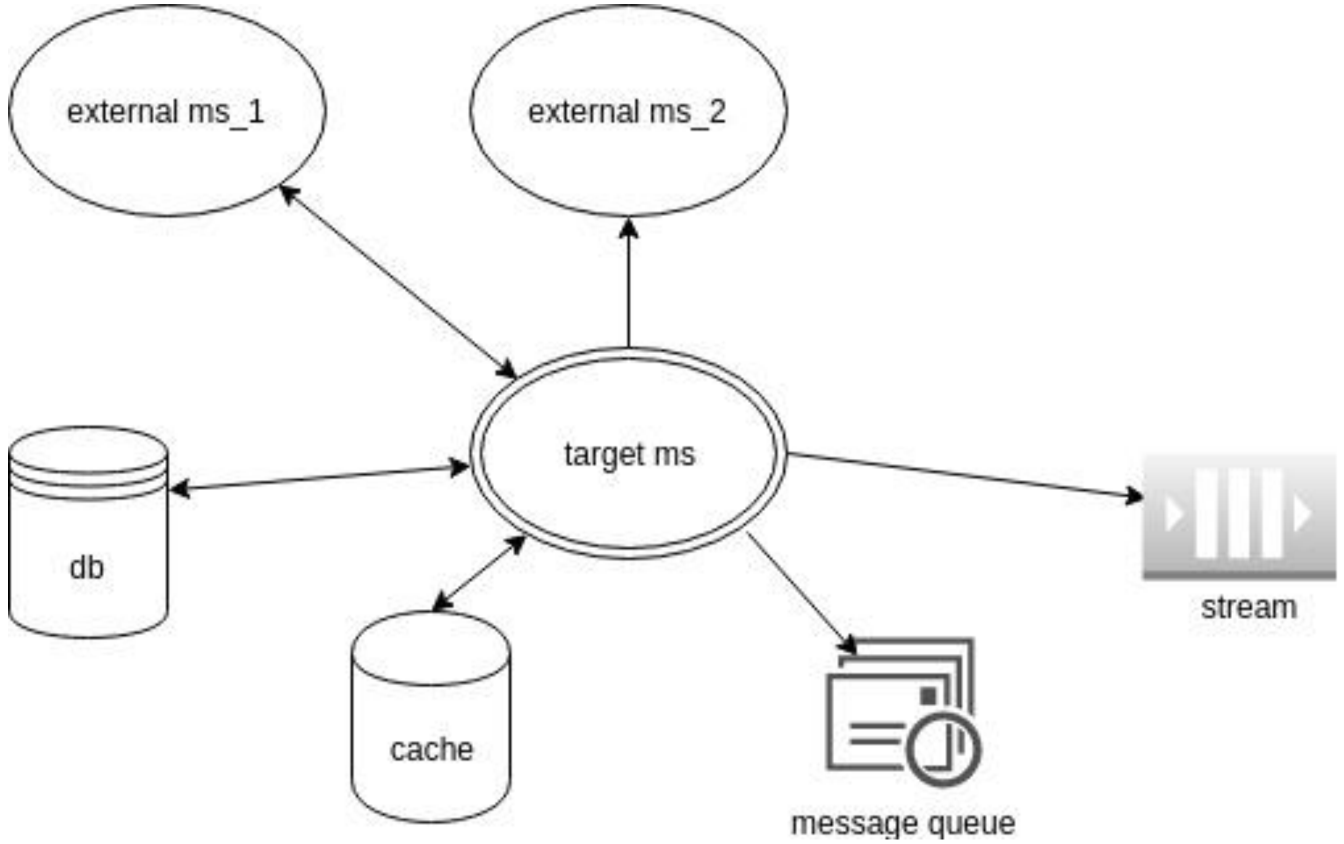
Mark the task as ready for deployment, monitoring if needed



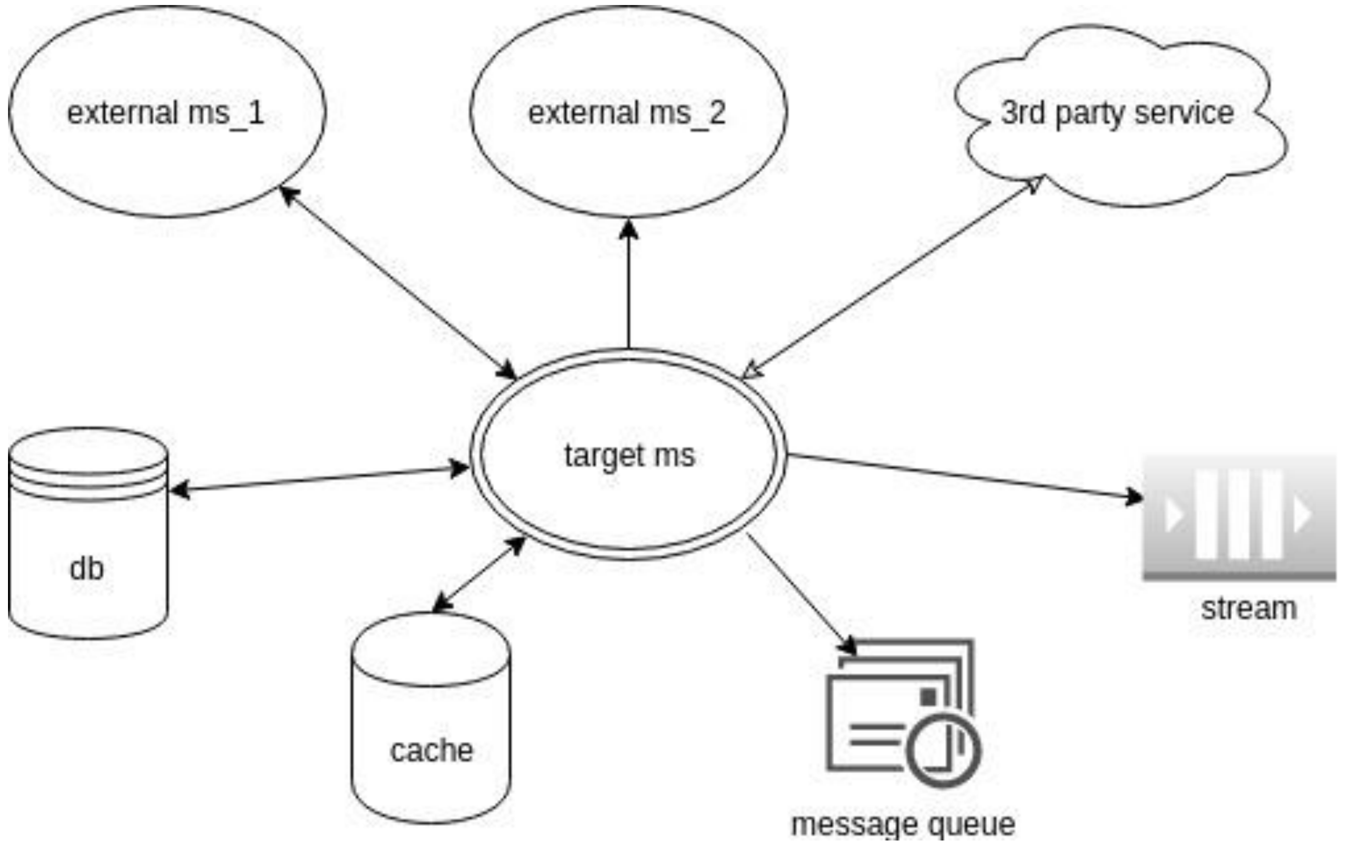
Approach to microservice testing



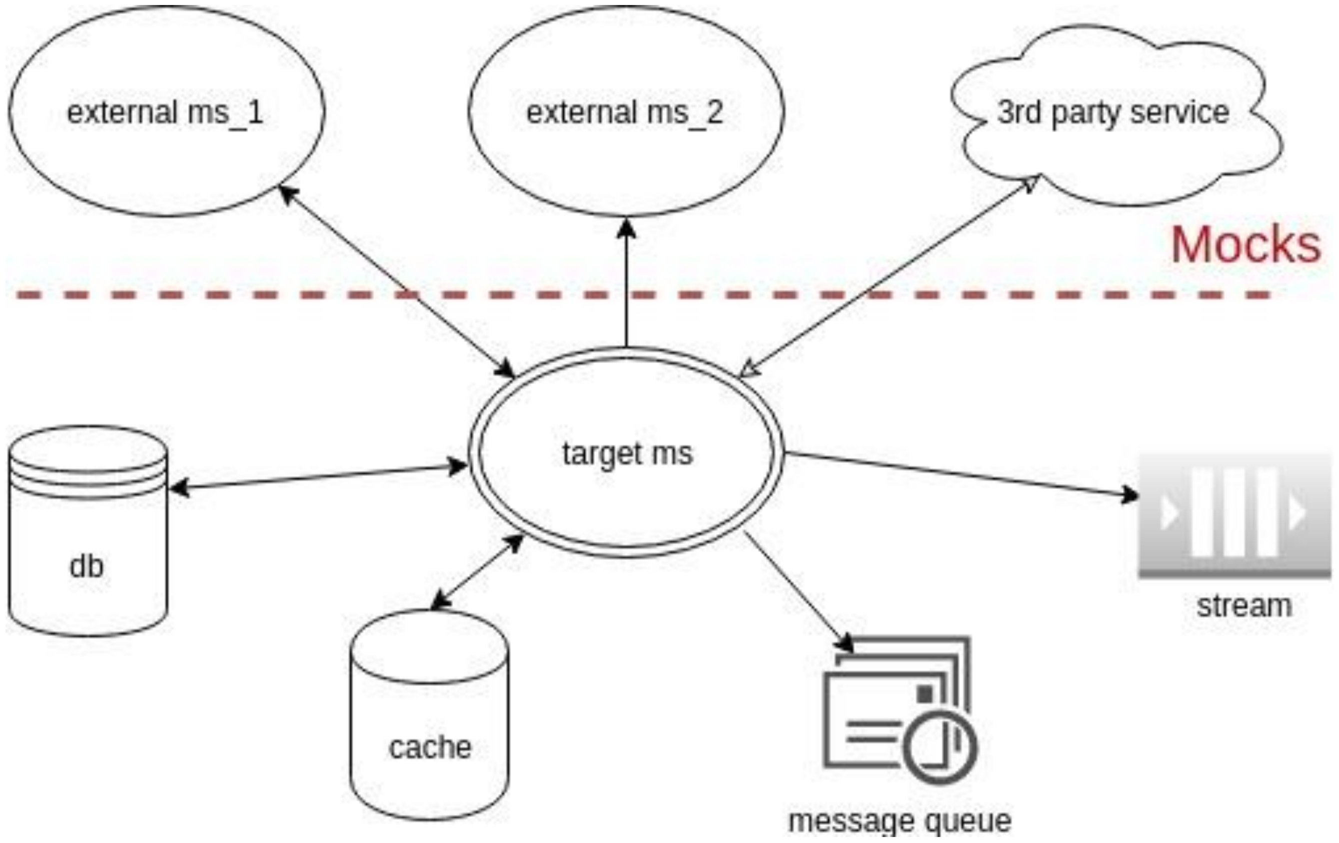
Approach to microservice testing



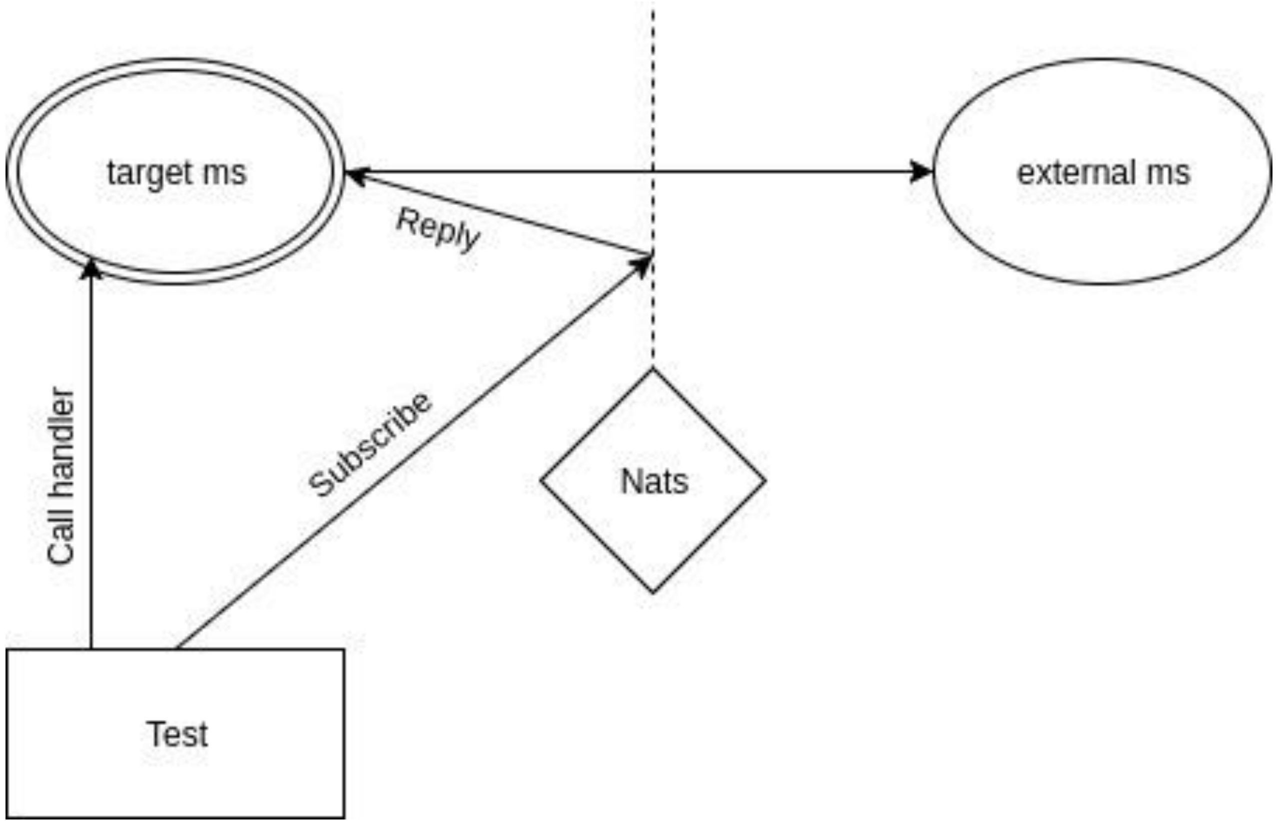
Approach to microservice testing

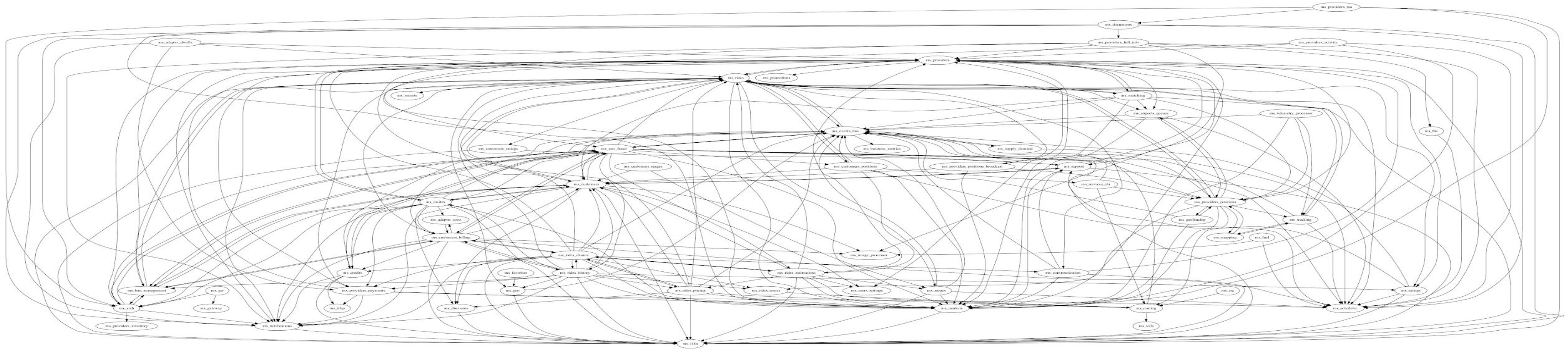


Approach to microservice testing

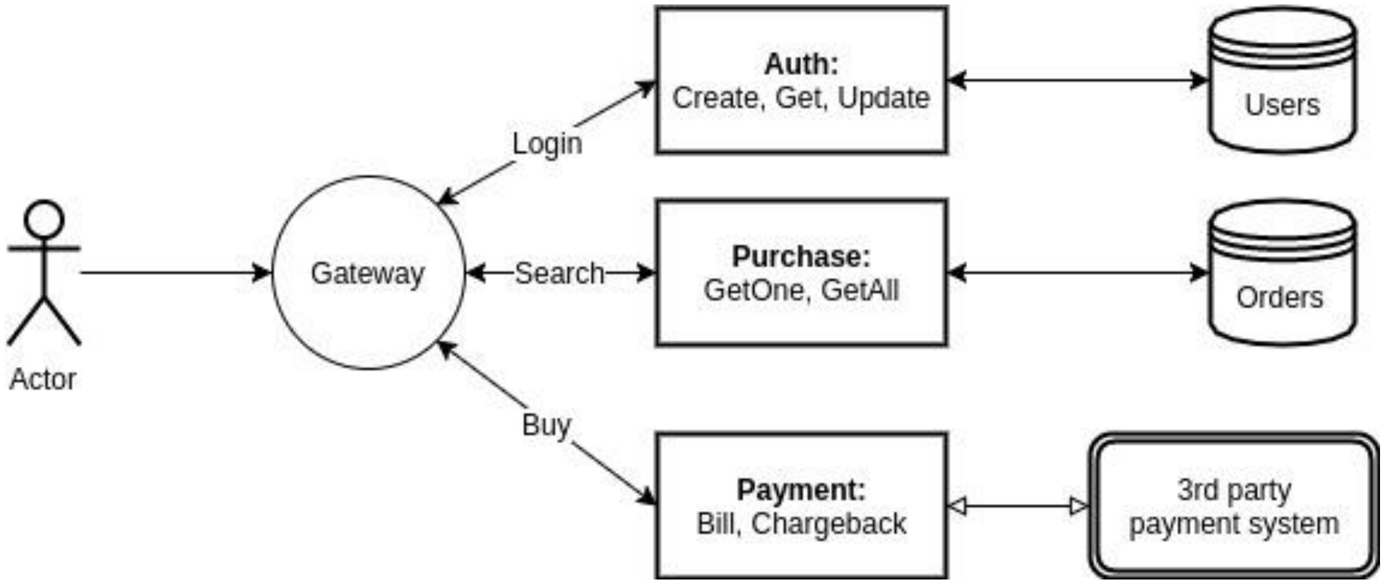


Mocks with external ms




ms_providers_rsu
ms_documents
ms_adapter_dwolla
ms_providers_full_info
ms_providers_activity
ms_providers
ms_rides
ms_promotions
ms_receipts
ms_matching
ms_airports_queues
ms_telemetry_processor
ms_fhv
ms_events_bus
ms_business_metrics
ms_supply_demand
ms_customers_ratings
ms_anti_fraud
ms_customers_surges
ms_customers_positions
ms_providers_positions_broadcast
ms_support
ms_services_eta
ms_customers
ms_invites
ms_providers_positions
ms_adapter_zooz
ms_geofencing
ms_tracking
ms_customers_billing
ms_snapping
ms_fuel
ms_image_processor
ms_rides_closure
ms_credits
ms_favorites
ms_rides_history
ms_rides_estimations
ms_communication
ms_ban_management
ms_gw
ms_providers_payments
ms_geo
ms_rides_pricing
ms_rides_routes
ms_users_settings
ms_surges
ms_eta
ms_ratings
ms_auth
ms_gateway
ms_ldap
ms_discounts
ms_markets
ms_routing
ms_scheduler
ms_providers_inventory
ms_notifications
ms_tolls
ms_i18n

Architecture example



Contract library

```
class AuthKeys(object):
  class CreateUserReq(object):
      FirstName = u'first_name'
      LastName = u'last_name'
  class CreateUserResp(object):
      UserId = u'user_id'

class AuthWrap(object):
  @staticmethod
  def create_user_req_wrap(first_name=None, last_name=None):
      keys = AuthKeys.CreateUserReq
      return {
          keys.FirstName: first_name or generate_string(),
          keys.LastName: last_name or generate_string(),
      }

  @staticmethod
  def create_user_resp_wrap(user_id=None):
      keys = AuthKeys.CreateUserResp
      return {
          keys.UserId: user_id or uuid.uuid4(),
      }
```

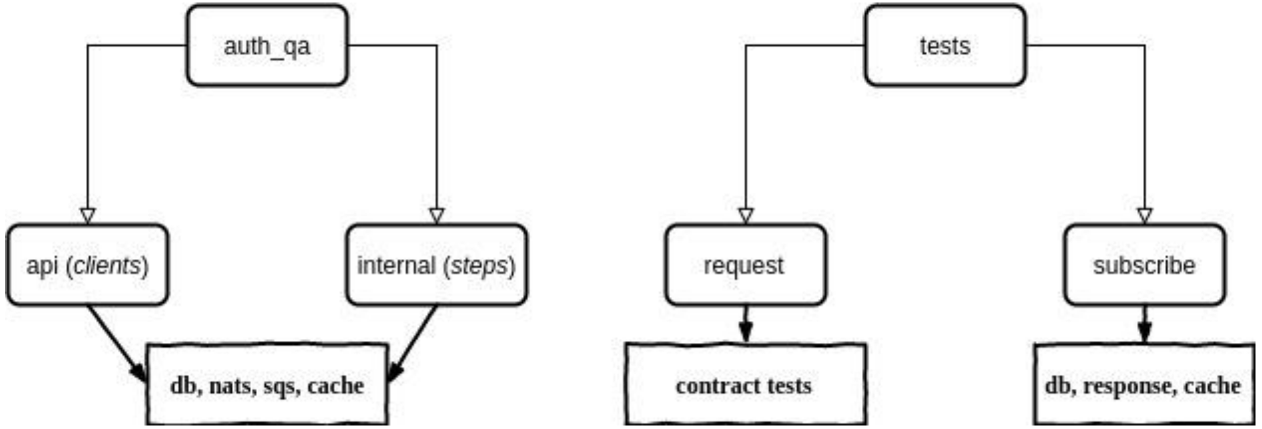
General library

```
def generate_string(chars):
  return u''.join(random.choice(chars) for _ in range(random.randint(6, 10)))


def get_current_time_utc():
   return datetime.datetime.utcnow()


def generate_ip():
   return u'.'.join(str(random.randint(0, 255)) for _ in range(4))
```

Test library

- **Steps decorator**
  - http
  - db
  - memcache
- **Fixtures**
  - positive (valid_string, valid_uuid)
  - negative (invalid_money, invalid_dictionary)
- **Tools**
  - dictionary comparer
  - microservice manager
  - default packages extensions

Project structure



Contract tests

1. Prepare data model
2. Subscribe on expected external ms subject
3. Async call handler (do not wait for response)
4. Wait for subscribed object called and catch it
5. Check request call

```python
def test_auth_get_users():
    # prepare data
    subcr_auth_get_user = subscr_steps.auth_get_user()
    msg_id = generate_uuid()
    user_id = generate_uuid()
    req_body = PaymentWrap.bill_req_wrap(user_id=user_id)
    # call handler
    ack_steps.bill(req_body, msg_id=msg_id)
    # assert
    subscr_steps.verify_auth_get_user_req(
        subcr_auth_get_user, msg_id=msg_id, user_id=user_id
    )
```

Behavior tests

1. Prepare data model
2. Subscribe on all external calls
3. Call handler synchronously
4. Catch and reply all external calls with mocks
5.
    a. Check response
    b. Check db record
    c. Check cache changes

```python
def test_create_user_response():
    # prepare data
    first_name = generate_string()
    last_name = generate_string()
    req_body = AuthWrap.create_user_req_wrap(
        first_name=first_name, last_name=last_name
    )
    # call handler
    response = req_steps.create_user(req_body)
    # verify response
    user_id = db_steps.get_user_id(first_name=first_name, last_name=last_name)
    resp_steps.verify_create_user_resp(response=response, user_id=user_id)
```

Why isolation?

- Fast multirun: **100k+** tests in **6 minutes**
- Fast test suite local run: **2k** tests in **1 min**
- Single ms should be compiled and run
- Easy to parallel in CI
- Contracts issues root cause analysis
- Run ms test suite for each commit

Why not API?

- Too slow
- Bad code coverage
- Full system should be run
- Expensive local hardware setup
- Difficult to test minor update in ms
- Not full control under the system

Honest?

- Each qa engineer knows all the contracts
- Good contract test coverage
- General lib for contracts wrappers
- Google do it in the same way

THANK YOU! :)

JUNO

@alex_chumakin
www.linkedin.com/in/achumakin

Useful links

Martin Fowler - Consumer-Driven Contracts

Jochen Wuttke - Building Test Infrastructure

Mike Wacker - End-to-End Tests