The first almost empty slide

The second almost empty slide

Enterprise-ready FastAPI

Alexander Ptakhin with ❤️ for PyBerlin 2024 Nov 27th

"Enterprise FastAPI" sounds weird

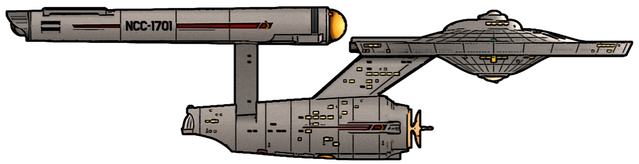
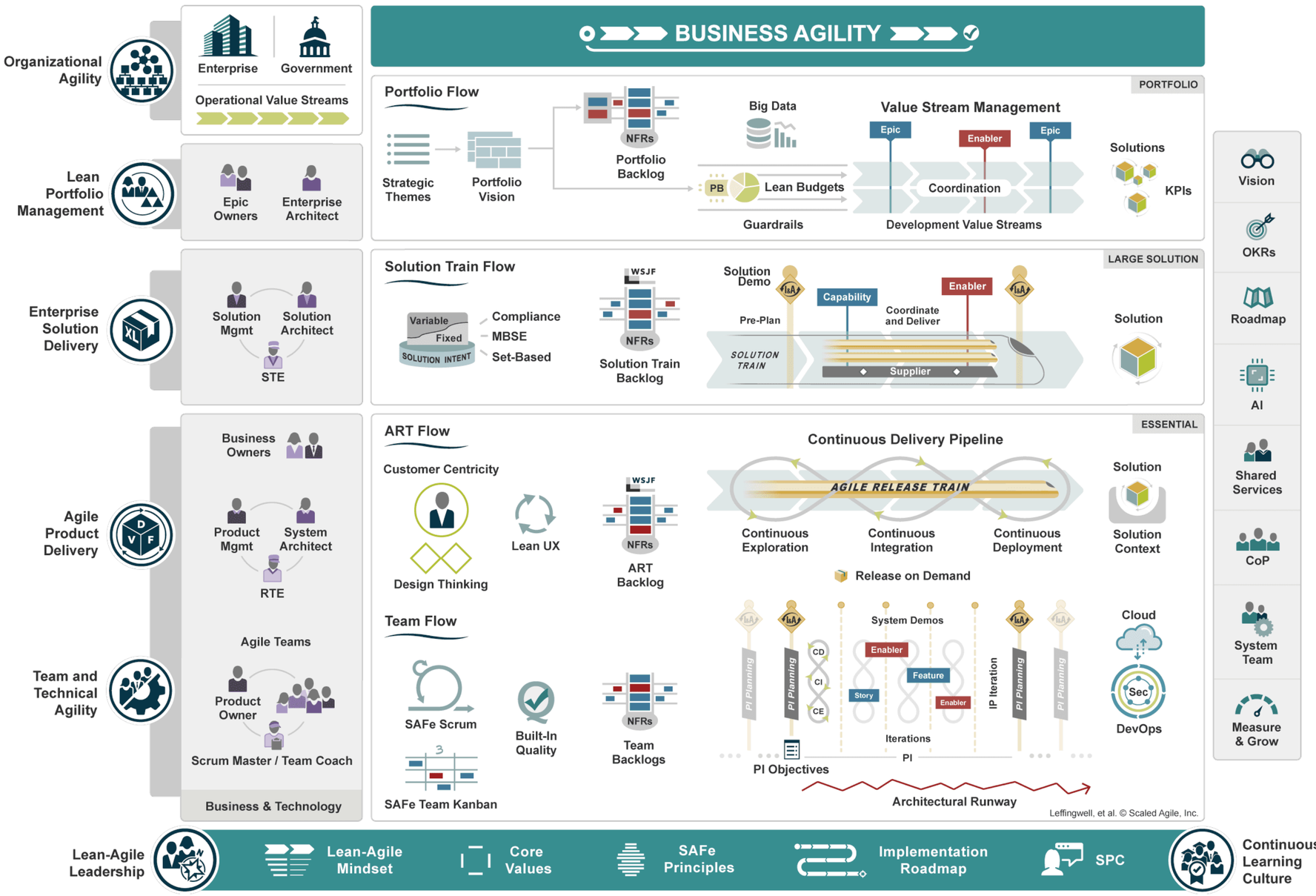
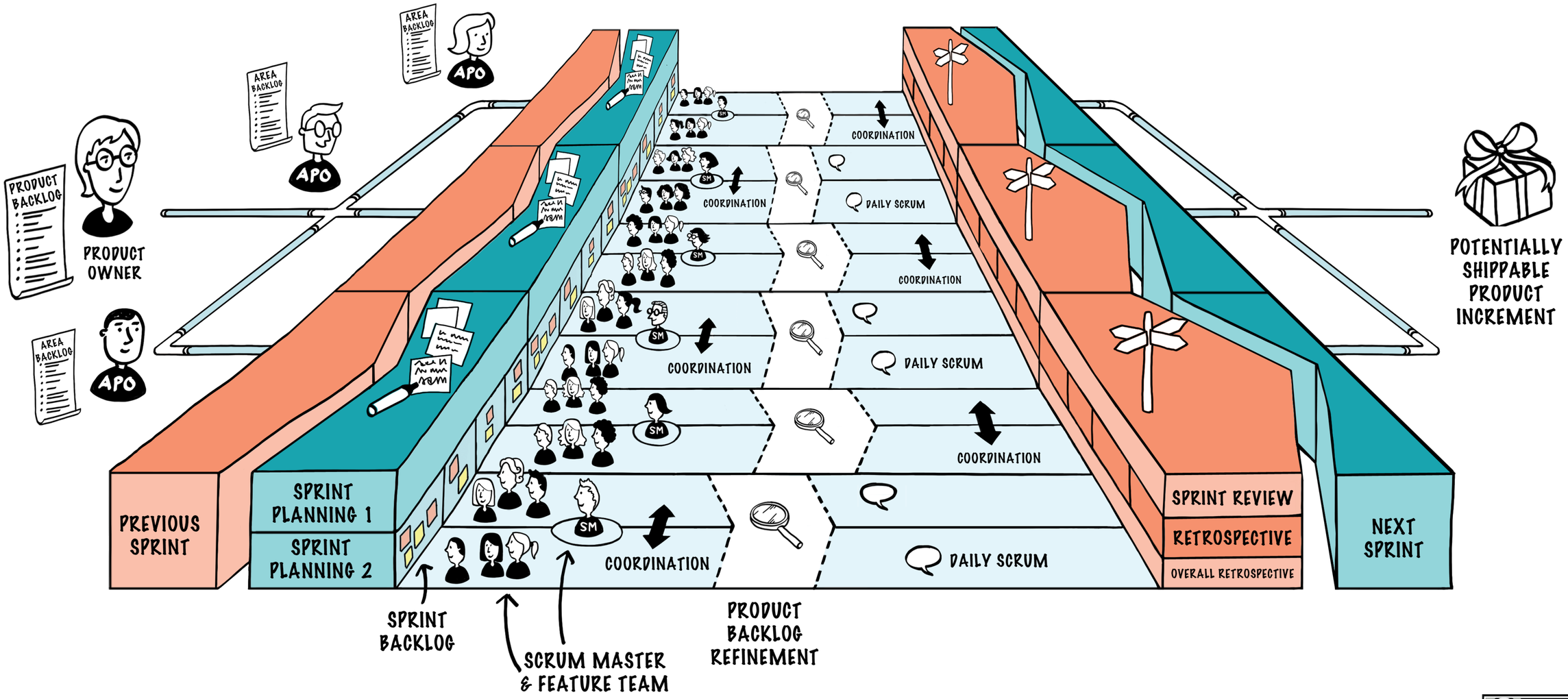


Image by Thomas B from Pixabay



SAfe https://scaledagileframework.com/#full

LeSS



http://less.works CC BY-ND

https://less.works/less/less-huge


Jira

It’s me!

Enterprise slide: why I'm the expert

- Alexander Ptakhin
- 10+ years with Python
- C++, Python 2, Python 3, JavaScript
- ~~not in production: Java, PHP, Ruby, Rust, C#~~
- Monoliths, distributed systems
- Not heavy-loaded
- I’m the tech lead of the team at Prestatech @ Berlin. We automate paperwork processing, helping to answer questions about giving credit or mortgages to businesses. It's not an enterprise yet, a startup

By MEDV

Short list of things to think about before start programming

Short list of things to think about

- Problem would be solved
- Architecture
- Teams
- Design
- Deployment
- Migrations
- Pipelines
- Environments

- Customers support
- Testing strategy
- Authentification, authorization
- Monitoring
- Development strategy
- Problem still would be solved

Long!

Boring!

~~Short~~ list of things to think about

- Problem would be solved
- Architecture
- Teams
- Design
- Deployment
- Migrations
- Pipelines
- Environments
- Customers support
- Testing strategy
- Authentification, authorization
- Monitoring
- Development strategy
- Problem still would be solve

Long!

Boring!

Wisdom or joy?

Enterprise – it's Java or C#

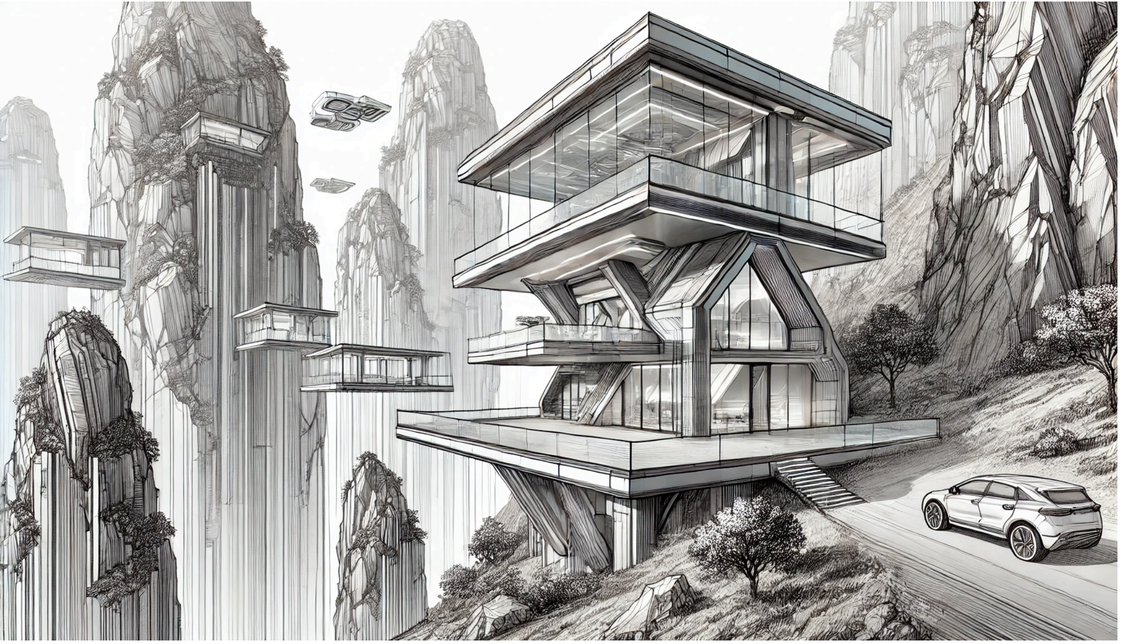
Image by GraphicMama-team from Pixabay

Wisdom and joy!

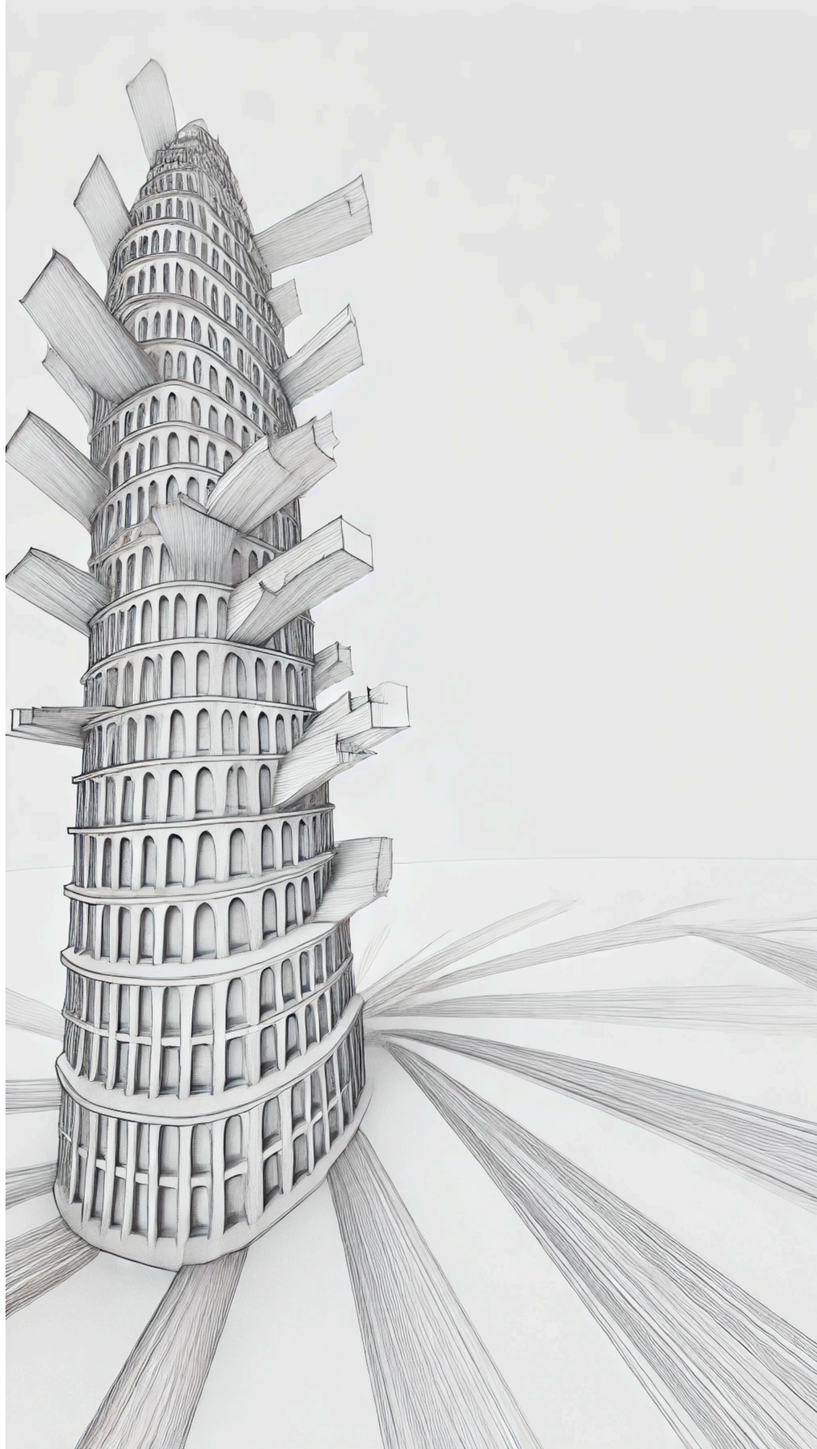
~~Short~~ list of things to think about

- Problem would be solved
- **Architecture**
- Teams
- Design
- **Deployment**
- **Migrations**
- **Pipelines**
- Environments
- Customers support
- **Testing strategy**
- Authentification, authorization
- **Monitoring**
- Development strategy
- Problem still would be solved

Let’s jump into the rabbit hole



Architecture



Context

- Around 60 services in production Python and C# serverless and Kubernetes
- New product, much frontend
- Not many users from the start
- We don't know Django
- We used Flask a bit
- Async FastAPI is fancier
  - Similar efforts with Flask, cooperative multitasking worth learning

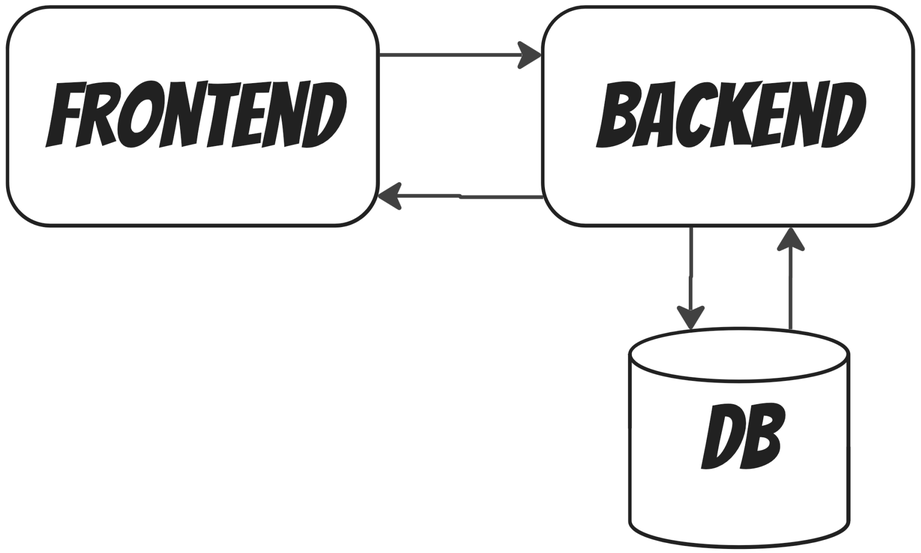
Architecture

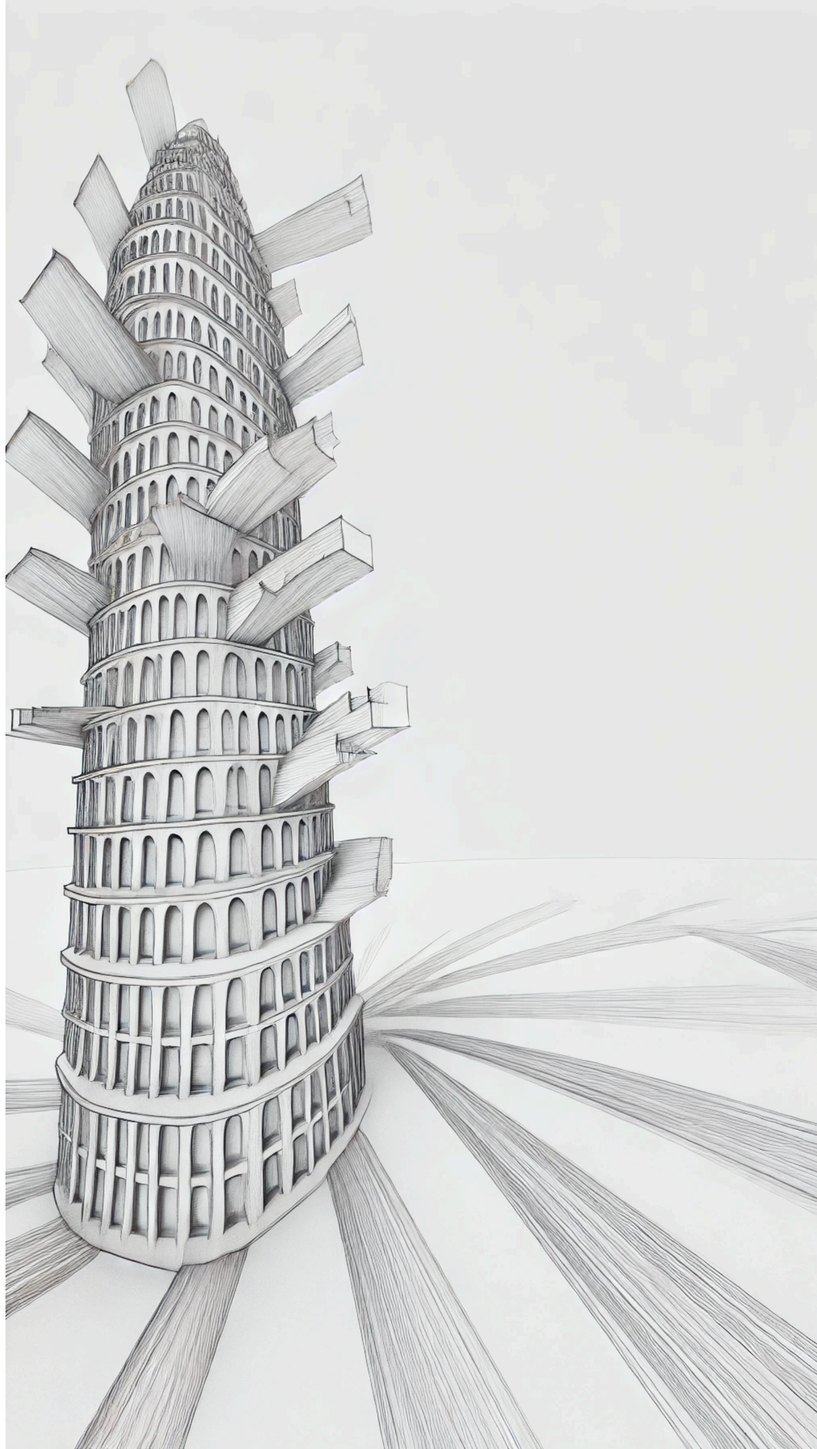
Choosing fancy one

- Monolith
- Microservices
- Modular monolith

FastAPI

- 3-tier. Make independent modules in the Backend we can move them out later

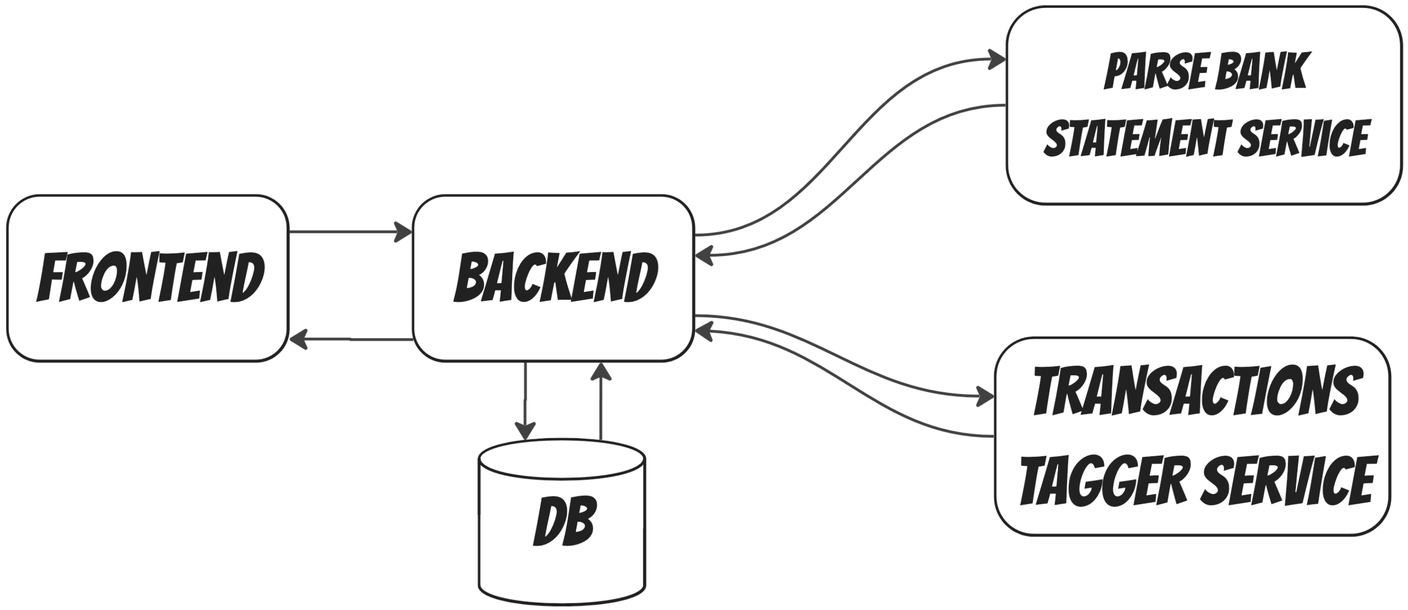


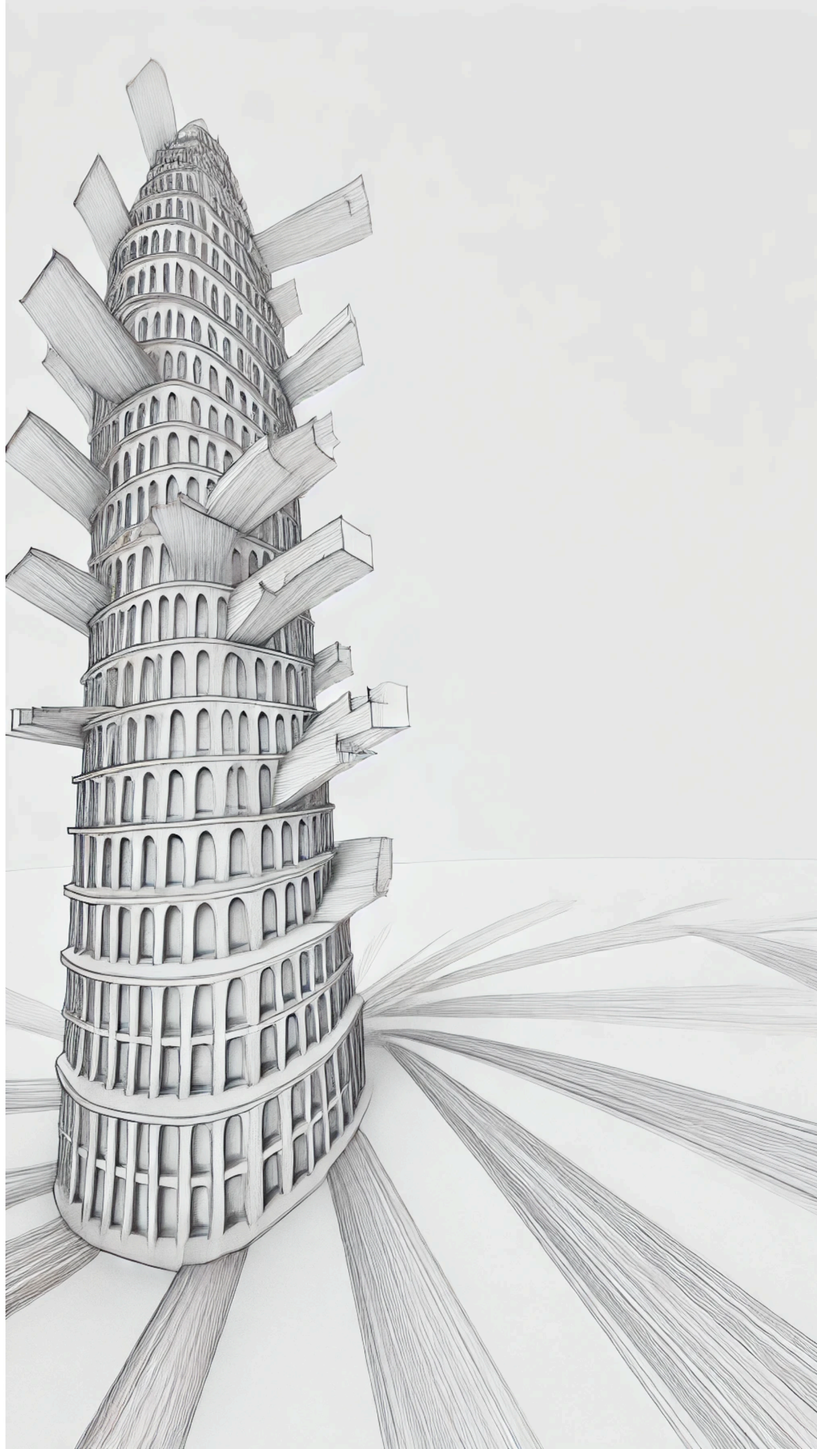


FastAPI

Folders

- And modules to support external parsing/tagging services





Testing strategy

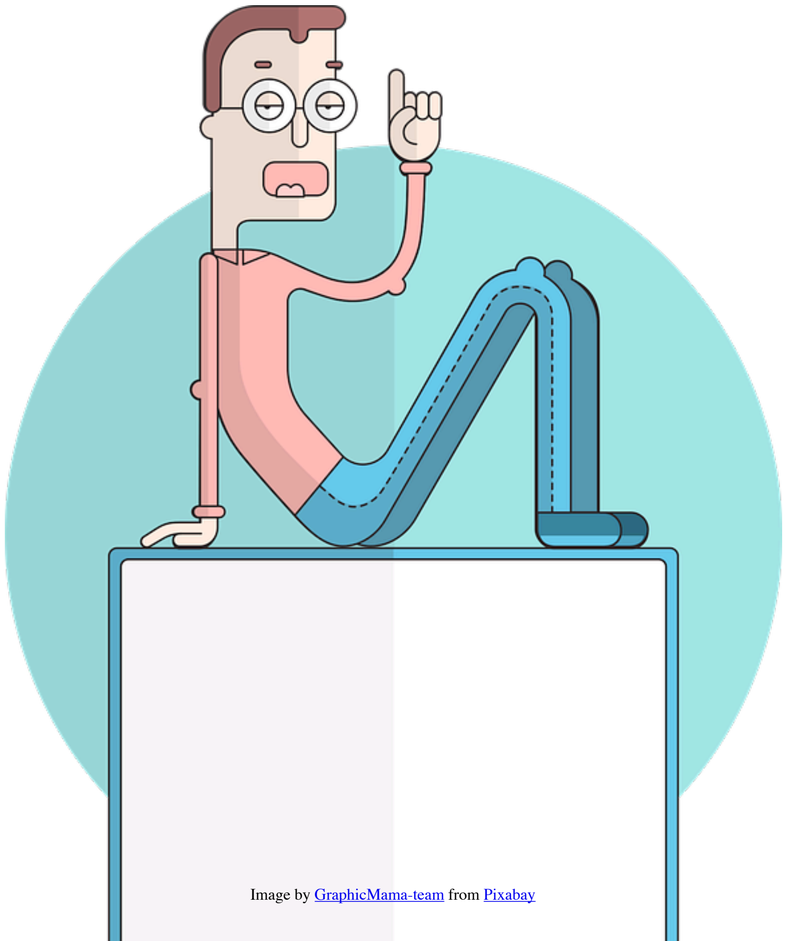


Image by GraphicMama-team from Pixabay

Tests

Testing strategy

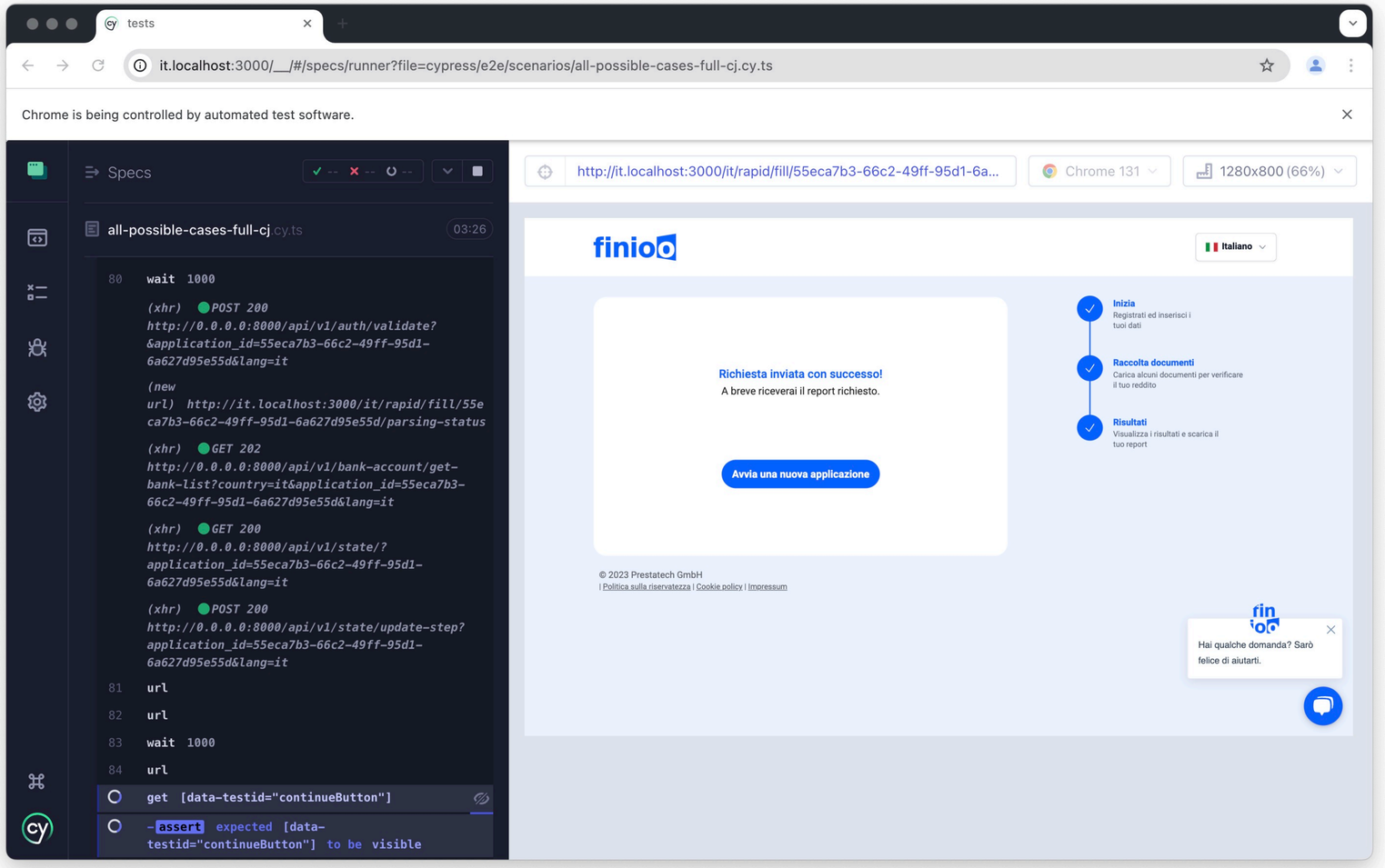
The testing strategy is a high-level understanding goals and approaches for testing product

- Testing business requirements implementation
- Acceptance – high level
- Unit – low level
- Manual

Tests

Acceptance (e2e) with JavaScript

- Cypress. If we are testing frontend, it is more straightforward to stick to JavaScript
- Even if some solution has Python scripts (e.g., Playwright)

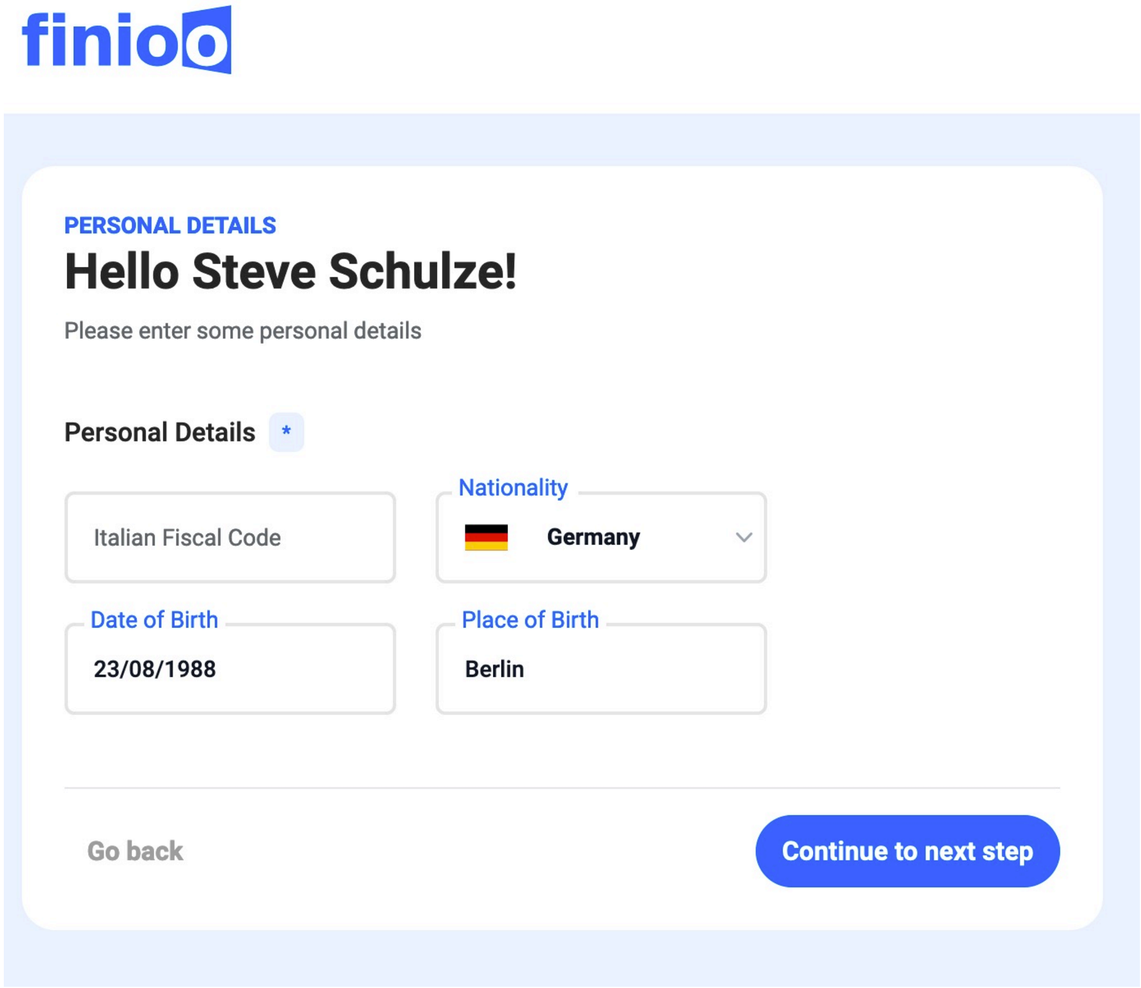


Tests

Acceptance (e2e) with JavaScript

```
let personalDetails = new
PersonalDetailsPage(region)

personalDetails.enterDateOfBirth(
  '23/08/1988')
personalDetails.enterPlaceOfBirth(
  ‘Berlin')
personalDetails.clickOnContinueButton()
```



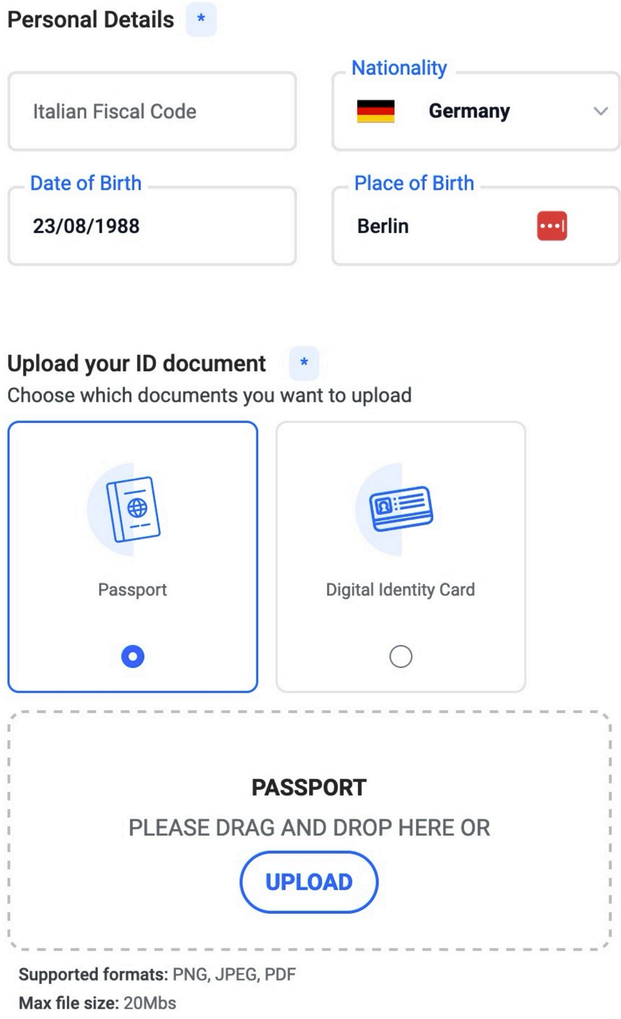
Tests

Acceptance (e2e) with JavaScript

```
let personalDetails = new
PersonalDetailsPage(region)

personalDetails.enterDateOfBirth(
    '23/08/1988')
personalDetails.enterPlaceOfBirth(
    'Berlin')
personalDetails.clickOnContinueButton()

personalDetails.chooseIdType(
    ‘passport')
personalDetails.uploadIdDocument(
    'resources/main-set/passport.jpg')
personalDetails.clickOnContinueButton()
waitUntilUrlChangesSeconds(
    ‘personal-details', 10)
```



Tests

Acceptance (e2e) with JavaScript

- Very expensive
- Very fragile
- Can cover green path scenario and that’s all

Image by Clker-Free-Vector-Images from Pixabay

We want to be sure we are not breaking API on lower level

Integration tests are lower-level part of acceptance tests

Tests

Integration example

```
@pytest.mark.require_db
@pytest.mark.asyncio
async def test_current_address_submit_successful(
    client: TestClient,
    valid_address_submit_request: AddressSubmitRequest,
    application: PreFilledApplication,
):
    …
```

Tests

Integration example

```
@pytest.mark.require_db
@pytest.mark.asyncio
async def test_current_address_submit_successful(
    client: TestClient,
    valid_address_submit_request: AddressSubmitRequest,
    application: PreFilledApplication,
):
    …
```

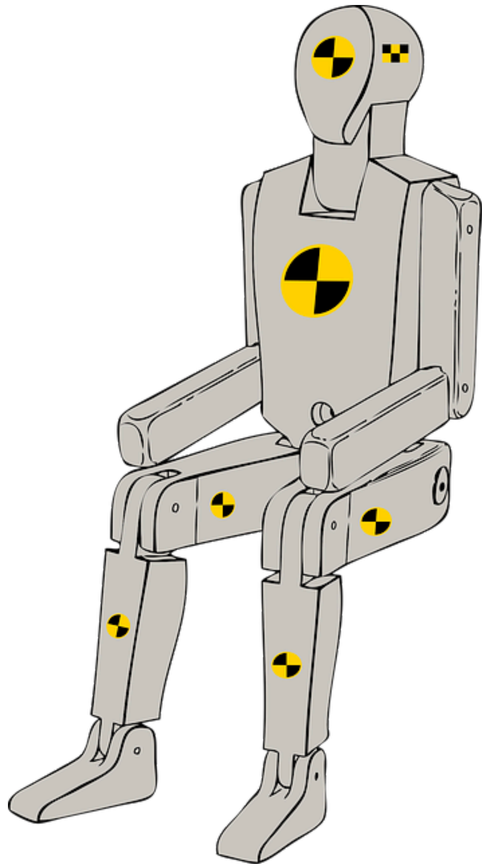


Image by OpenClipart-Vectors from Pixabay

Tests

Integration example

```
@pytest.fixture
def valid_address_submit_request() -> AddressSubmitRequest:
    return AddressSubmitRequest(
        street='Musterstraße',
        houseNum='11',
        postcode='12345',
        city='Musterstadt',
    )
```

Image by Sebastian Nikiel from Pixabay

Tests

Integration example

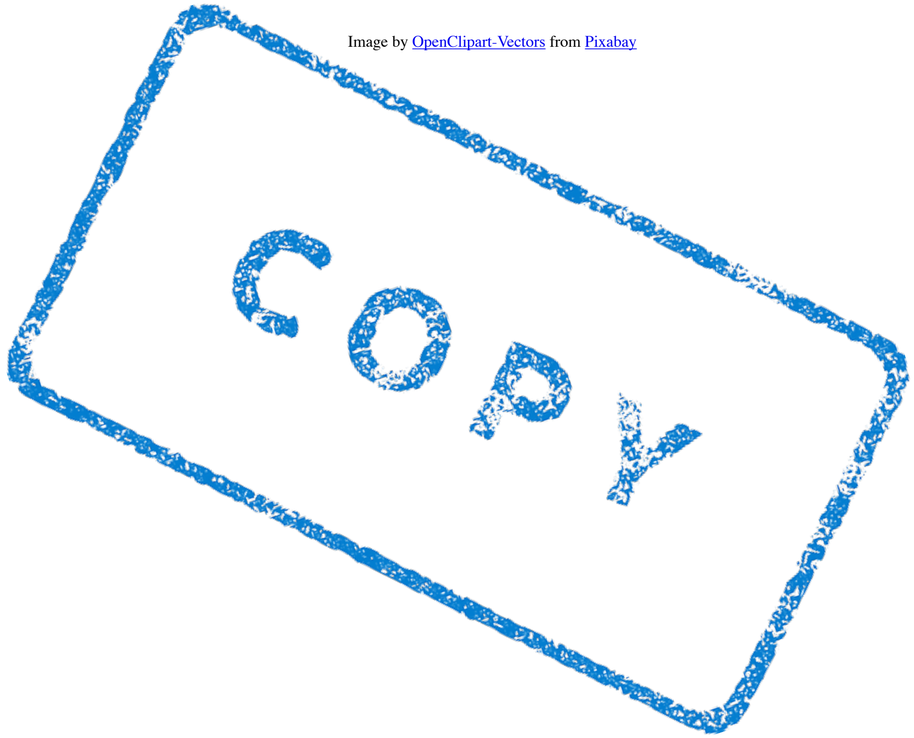


Image by OpenClipart-Vectors from Pixabay

```python
class PreFilledApplication(BaseModel):
    user_id: UUID
    user_token: str
    application_id: UUID

@pytest.fixture
async def user_with_application(db: Database) -> PreFilledApplication:
    ...
    return PreFilledApplication(
        user_id=...,
        user_token=...,
        application_id=...,
    )
```

mage by Clker-Free-Vector-Images from Pixabay

Tests

Integration example

```
response = client.post(
    '/api/v1/current-address/submit',
    params={'application_id': application.application_id},
    json=valid_address_submit_request.model_dump(by_alias=True),
    headers={'Authorization': f'Bearer {application.user_token}'},
)

assert response.status_code == status.HTTP_201_CREATED, response.text
assert response.json()['success'], response.text
```

Tests

Unit

- Test business requirements, not internals
- Very fast local

Image by OpenClipart-Vectors from Pixabay

Tests

Unit

```
from freezegun import freeze_time

@freeze_time('2023-08-21')
def test_fiscal_doc_last_year():
    assert (
        get_fiscal_doc_recency_error({'form_year': '2022'}) is None
    )

@freeze_time('2023-08-21')
def test_fiscal_doc_old_year():
    result = get_fiscal_doc_recency_error({'form_year': ‘2021'})
    assert result[1] == 'recency'
```

Tests ↗ / Quality ↗ / Joy ↗

Depends or not Depends?

Depends

Dependency override

```
class Emailer:
    async def send_email(self, email: str, ...) -> None:
        ...

def get_emailer(
    settings: Settings = Depends(get_settings),
) -> Iterator[Emailer]:
    yield Emailer(
        api_key=settings.sendgrid_api_key,
        default_from_email=settings.sendgrid_from_email,
    )
```

Depends

Dependency override

```
@router.post('/send-invite')
async def send_invite(
    application_id: UUID,
    emailer: Emailer = Depends(get_emailer),
) -> JSONResponse:
    ...
    await emailer.send_email(
        to_email=...,
        subject=...,
        content=...,
    )
```

Depends

Dependency override

```
from unittest.mock import create_autospec

@pytest.fixture()
def emailer_mock() -> Iterator[Emailer]:
    mock = create_autospec(Emailer)
    app.dependency_overrides[get_emailer] = lambda: mock
    yield mock
    del app.dependency_overrides[get_emailer]
```

Depends

Dependency injection

```
import punq

container = punq.Container()

def get_emailer() -> Emailer:
    return container.resolve(Emailer)
```

Depends

Dependency injection

```
@pytest.fixture()
def emailer_mock() -> Iterator[Emailer]:
    emailer = create_autospec(Emailer)
    container.register(Emailer, instance=emailer)
    yield emailer
    container.register(Emailer, instance=None)
```

Depends

Dependency injection

```
from inject import container

@asynccontextmanager
async def lifespan(app: FastAPI):
    settings: Settings = get_settings()
    emailer = Emailer(
        api_key=settings.sendgrid_api_key,
        default_from_email=settings.sendgrid_from_email,
    )
    container.register(Emailer, instance=emailer)
    yield
```

Depends

Dependency injection

- More symmetrical usage of registering and using
- Long-living objects instances creation

Depends and Depends + DI!

Configuration ease ↗ / Joy ↗

API Versioning

- Backward compatibility if we need it
- Backend for frontend: sometimes just refresh the page
- But not with public client API. Not with mobile devices also
- Publish new version, keep old compatible

API Versioning

- There are many unsafe changes:
  - Adding optional parameters to endpoints
  - Removing parameters from endpoints
  - Speeding up the endpoint
- [safe] Add a new endpoint no one uses yet

API Versioning

- Per router
  - Because it's often the prefix /api/v1/
- Per endpoint
  - More flexible
    /api/v1/create-application
    /api/v2/create-application

Simpler changes ↗ / Joy ↗

Migrations

Image by Gordon Johnson from Pixabay

Migrations

Choosing database management system

- Relations
- Transactions
- JSON
- Simple analytics on top with any viewer
- Many PostgreSQL instances already
- PostgreSQL

Image by Jose Alexis Correa Valencia from Pixabay

Migrations
Context

- We have too few data
- We are keeping backward-compatible
  - A few times, custom scripts to migrate existing data to the new structure
- Apply migrations in the pipeline before the release
- Otherwise, If we have many not migrated database instances
  - We have a difficult situation

Migrations
Difficult case

- New code should support old data schema
- Old code should support new data schema

Migrations

Difficult case

```
# added delete_reason column to application table
# but didn't migrate this database instance
stmt = select(application_table).where(
    application_table.c.id == new_id,
)
result = await conn.execute(stmt)
```

Migrations

Difficult case

```
# added delete_reason column to application table
# but didn't migrate this database instance
stmt = select(application_table).where(
    application_table.c.id == new_id,
)
result = await conn.execute(stmt)

# ---
E sqlalchemy.exc.ProgrammingError:
(sqlalchemy.dialects.postgresql.asyncpg.ProgrammingError)
<class 'asyncpg.exceptions.UndefinedColumnError'>: column
application.delete_reason does not exist
```

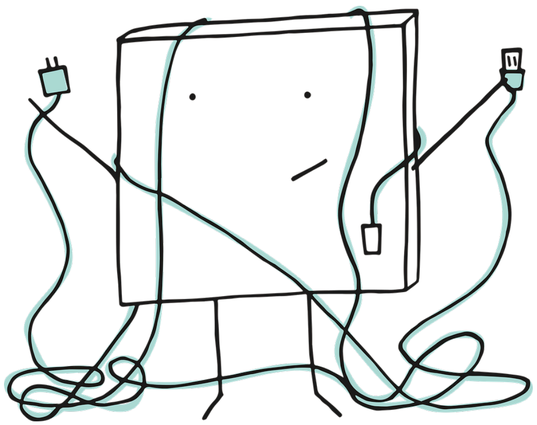


Image by Manfred Steger from Pixabay

Migrations

The fix

```
# make an explicit query list
# for different versions
stmt = select(
    application_table.c.id,
    application_table.c.user_id, …
).where(
    application_table.c.id == new_id,
)
result = await conn.execute(stmt)
```

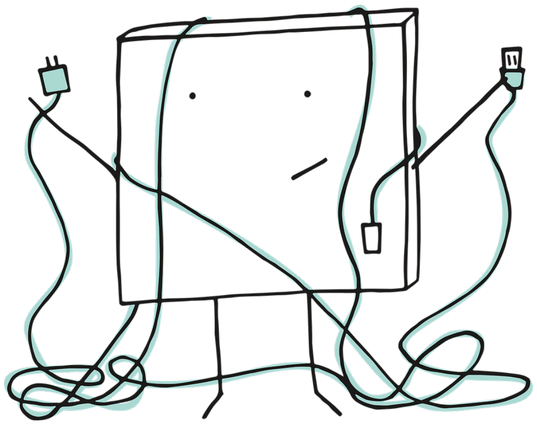


Image by Manfred Steger from Pixabay

Migration work ↗ / Joy →

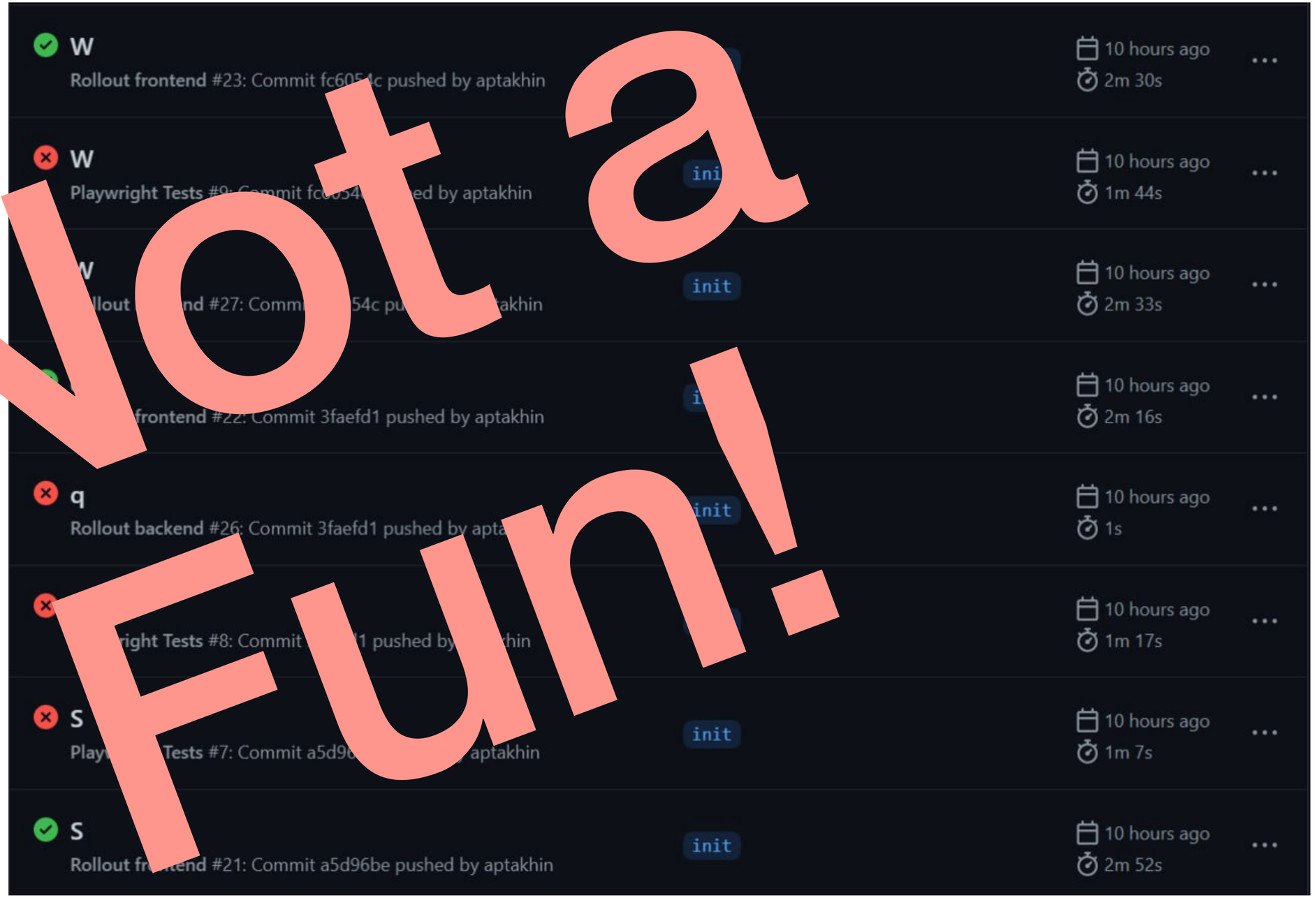
Pipelines

We understand what and why we automate

YAML

Pipelines

YAML



Not a Fun!

Pipelines
YAML

- Fragile to edit
- Can’t test locally (in most cases)

Pipeline
Advices

- No YAML-spaghetti!
- bash/python
  - Run bash or python
  - Test locally
  - Pass parameters to run scripts in YAML
- YAML
  - Write as little as possible

pre-commit

Local githooks on every commit

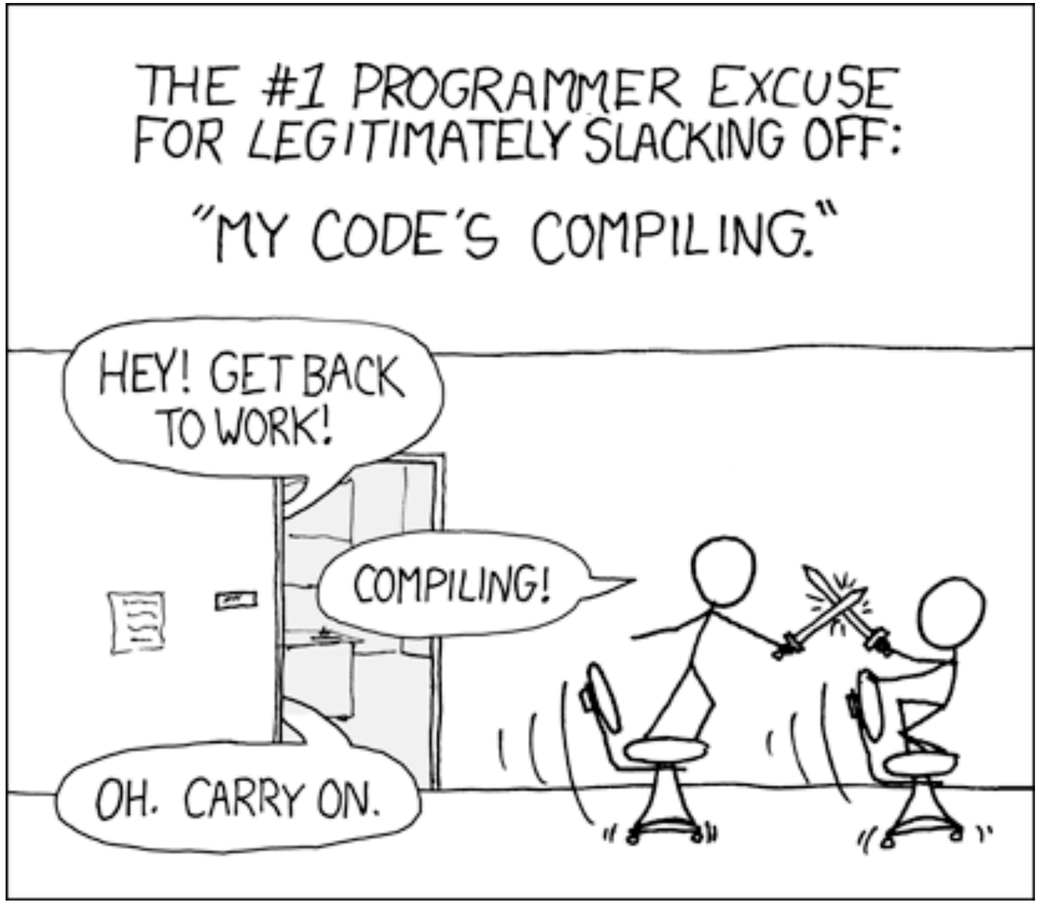
- check valid YAML
- ruff format
- ruff check --fix
- pytest -m unit
- Overall it should be fast, otherwise Joy ↘
- No more comments about style in pull requests / Joy ↗
- Less context switching on pull requests checks / Joy ↗

Style ↗ / Quality ↗ / Joy ↗

Pipelines

Unsolved problem: long builds

- We had a scripting language
- Long building Docker images of Python application in pipelines
- This is an unsolved XXI century problem 🙃



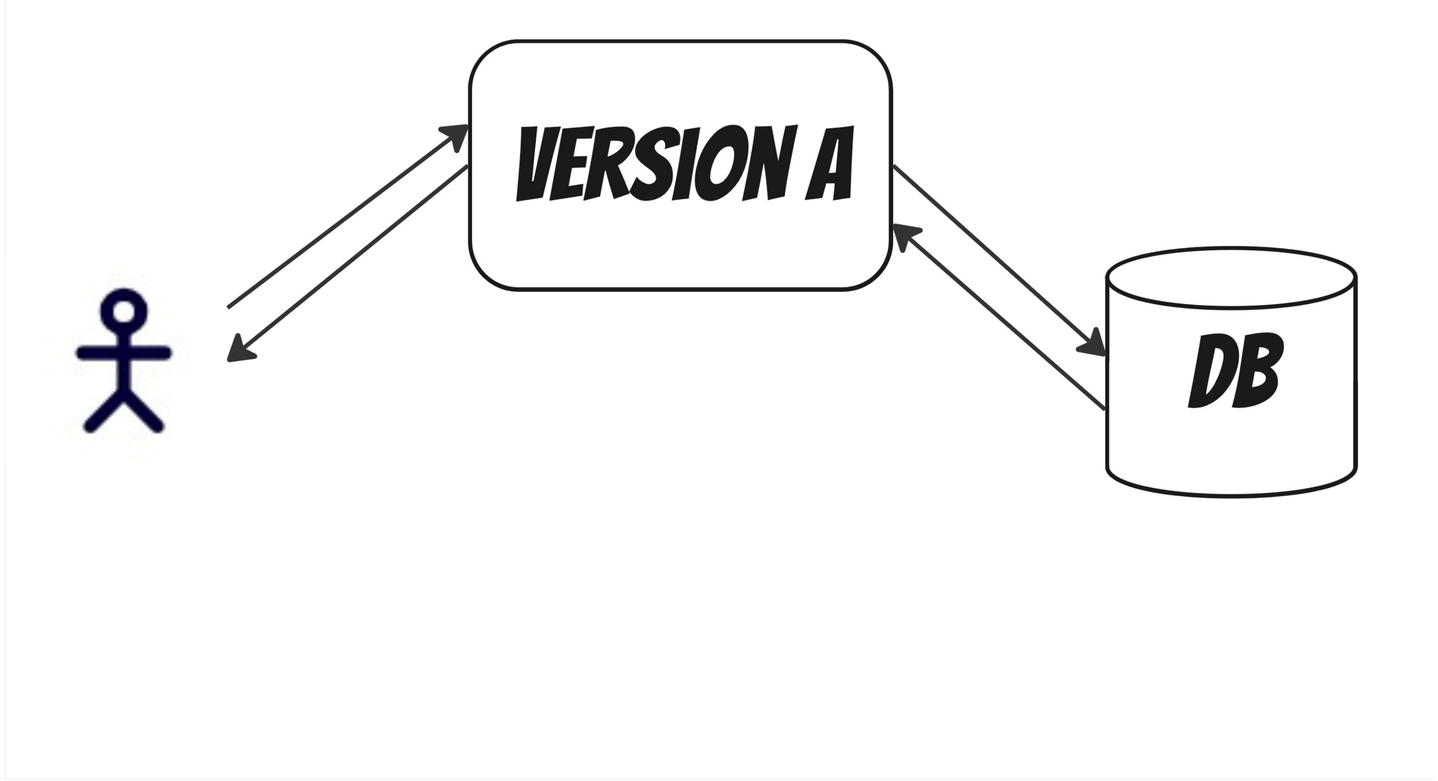
https://xkcd.com/303/

Deployment strategies

Have an opportunity to deploy
Have an opportunity to rollback with minimal damage

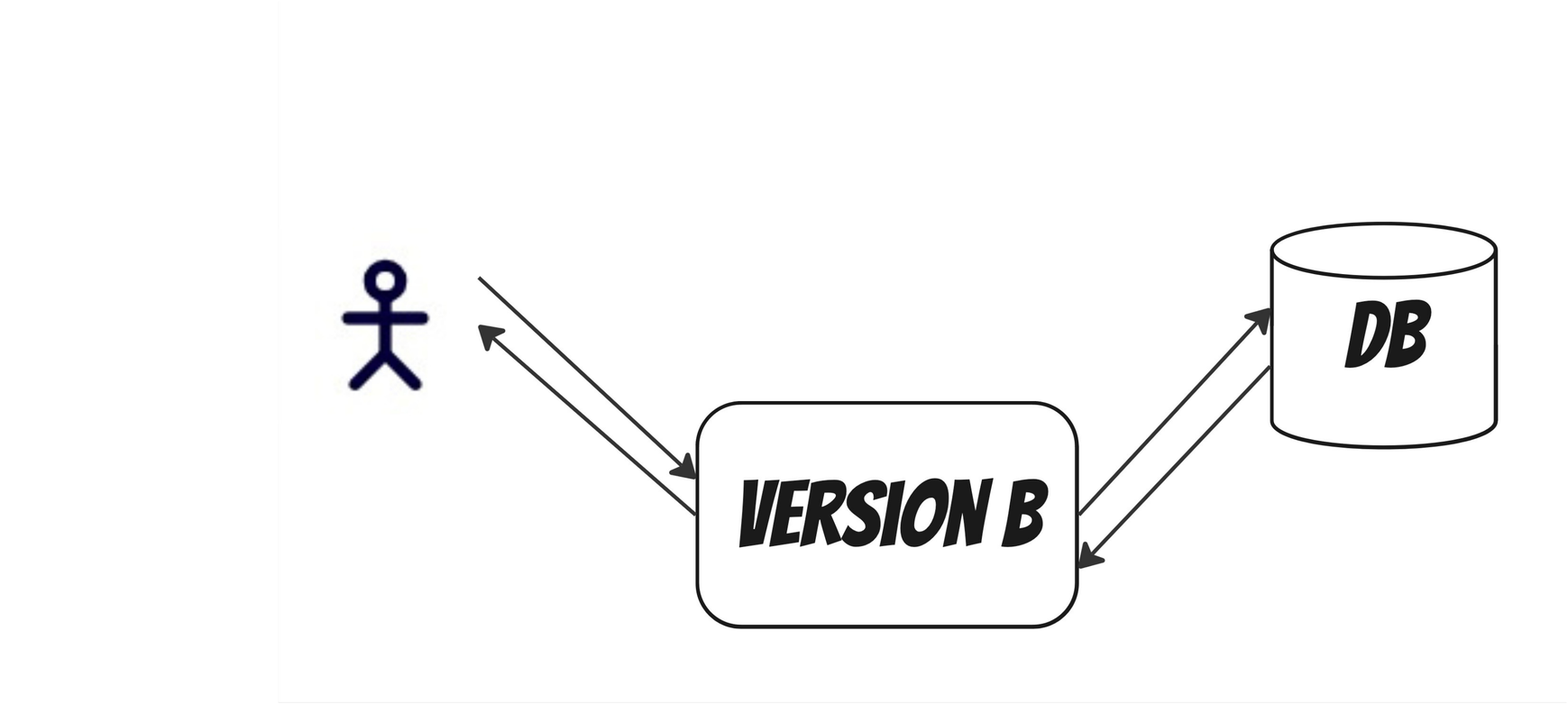
Deployment strategies

Recreate or in-place



Deployment strategies

Recreate or in-place



Deployment strategies

Recreate or in-place

- Pros: simple
- Cons: downtime
- Cons: unexpectable downtime to fix

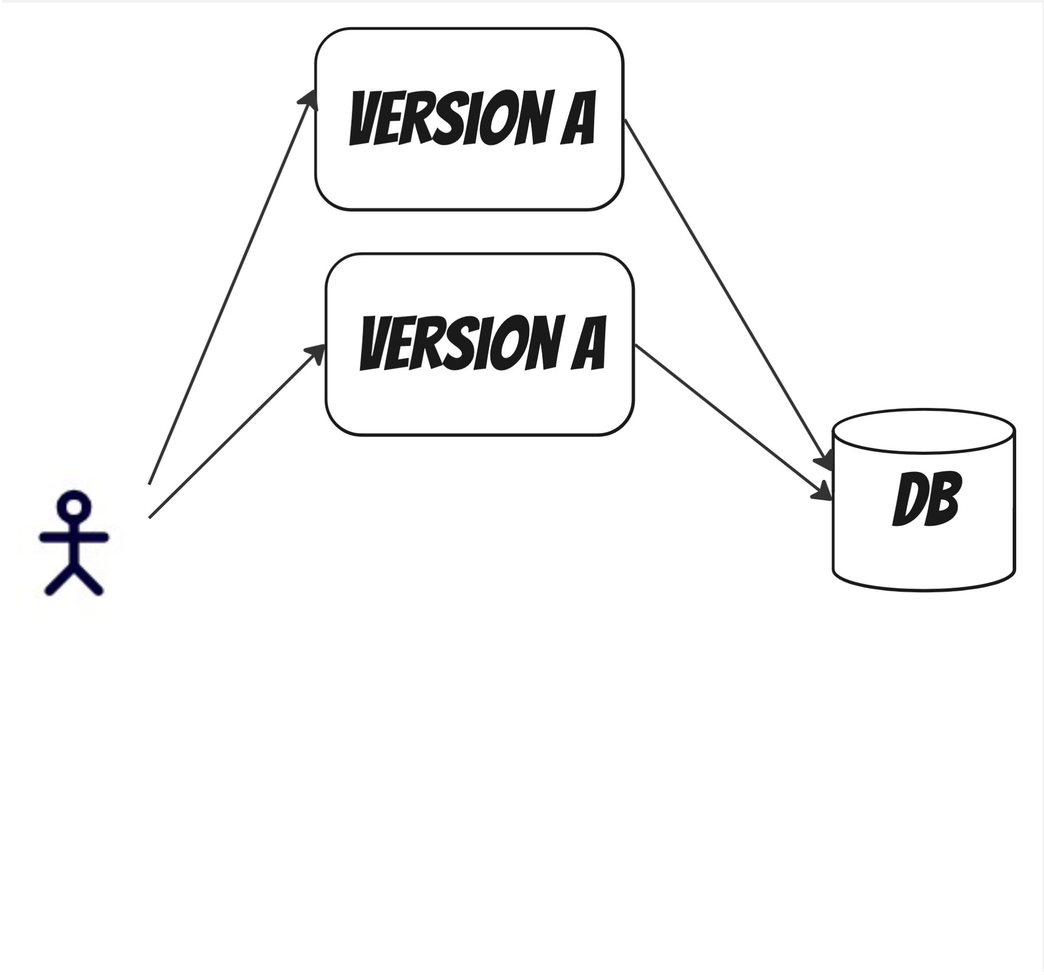
Deployment strategies

Ramped or rolling-update or incremental or canary

Version B is slowly rolled out and replacing version A

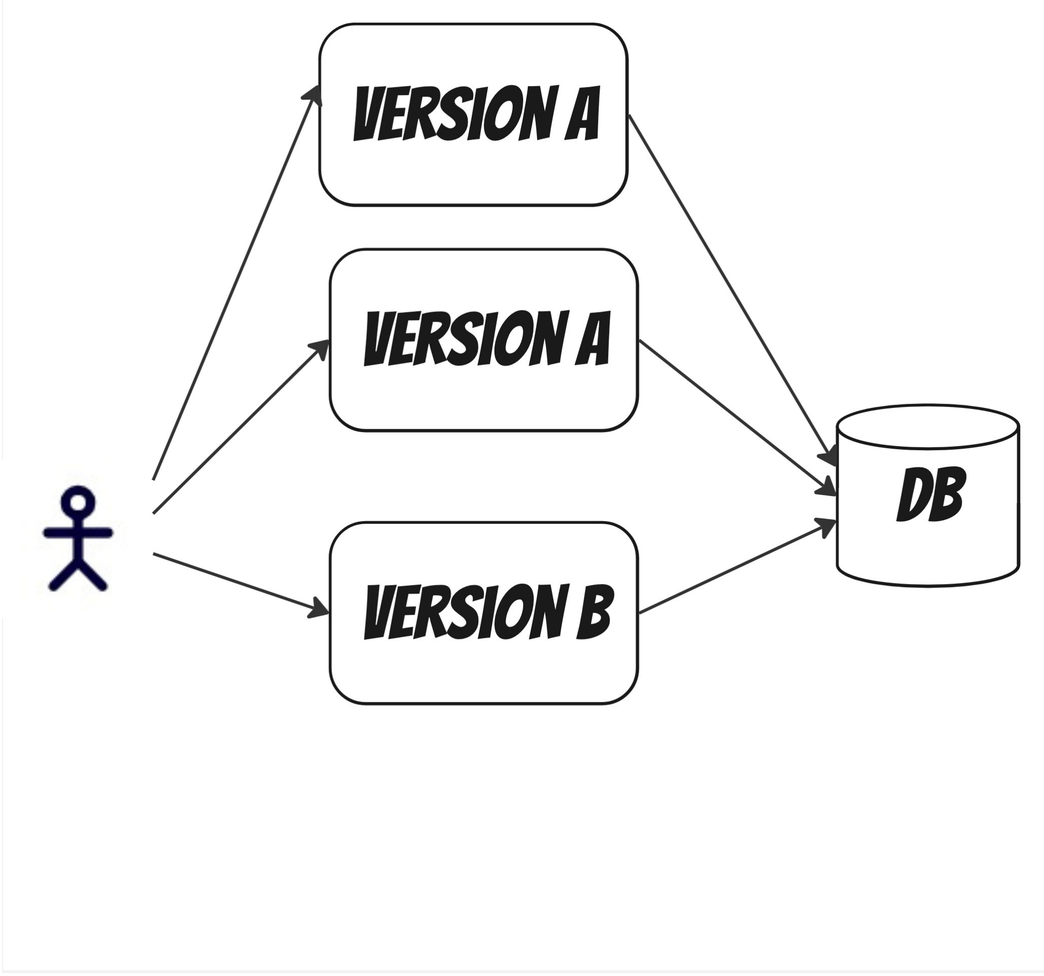
Deployment strategies

Ramped or rolling-update or incremental or canary



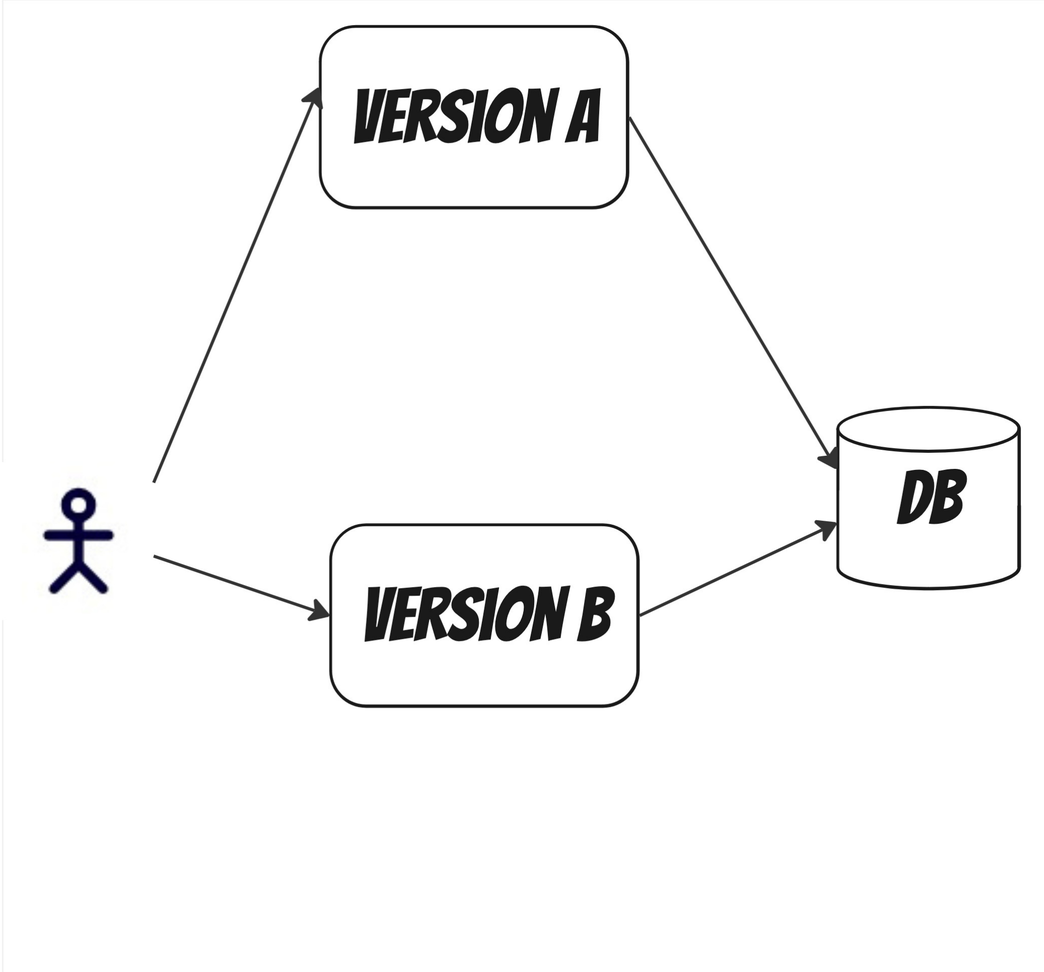
Deployment strategies

Ramped or rolling-update or incremental or canary



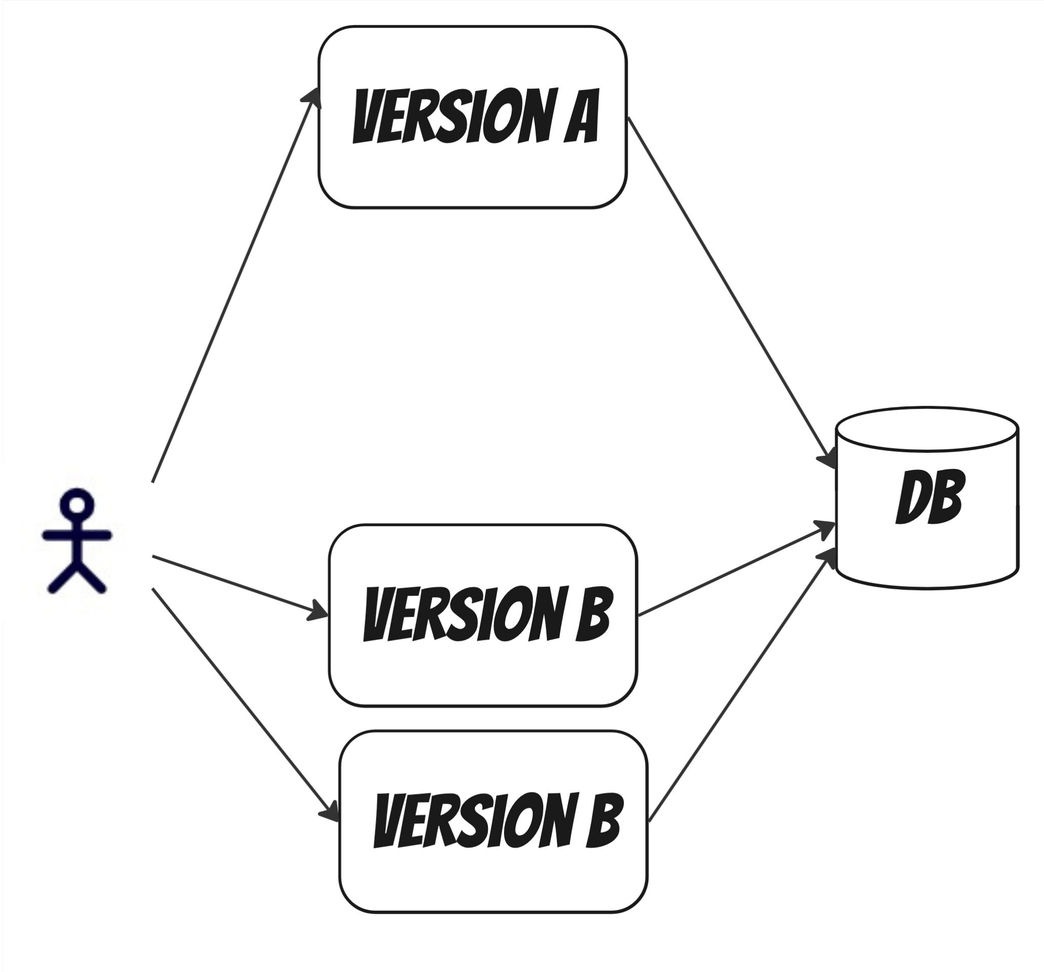
Deployment strategies

Ramped or rolling-update or incremental or canary



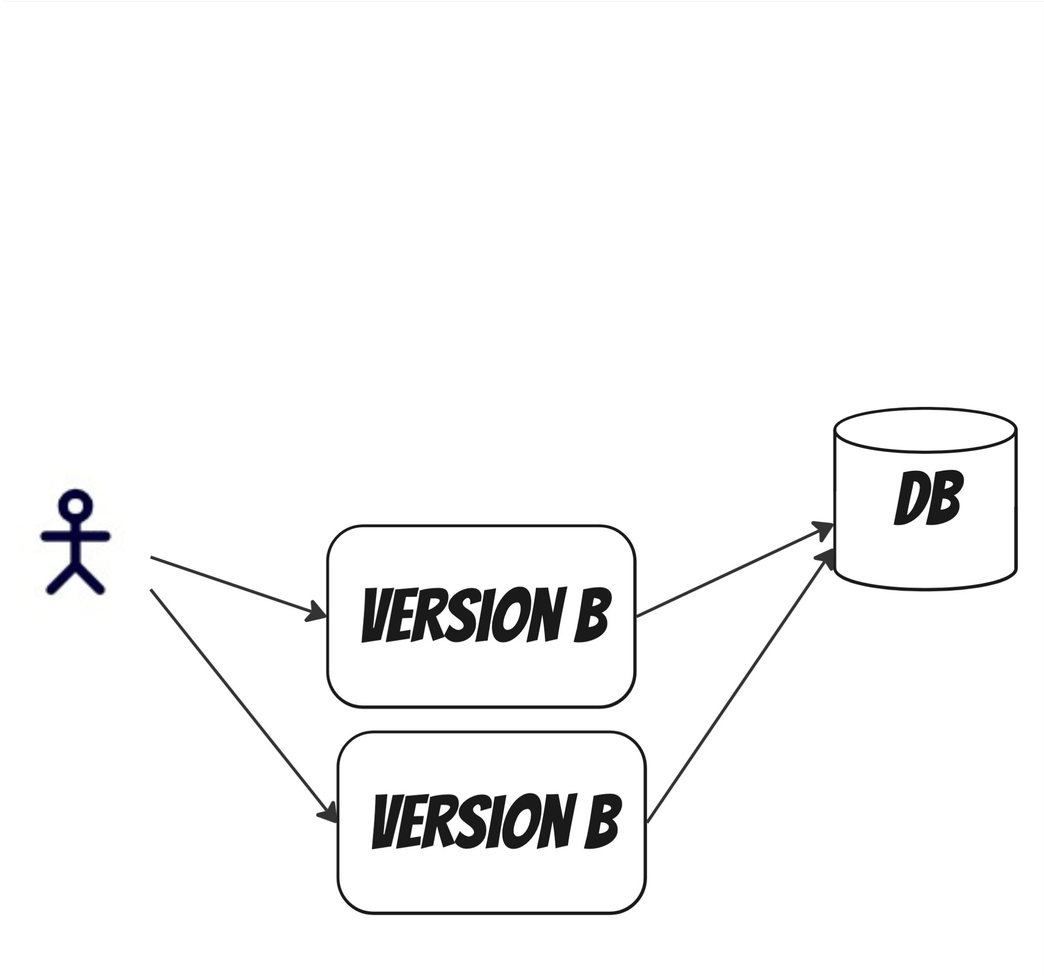
Deployment strategies

Ramped or rolling-update or incremental or canary



Deployment strategies

Ramped or rolling-update or incremental or canary



Deployment strategies

Ramped or rolling-update or incremental or canary

- Pros: no downtime
- Cons: need to implement it manually or have an orchestrator for it

Deployment strategies

One line about others

- Blue/Green: Version B is released alongside version A, then the traffic is switched to version B, apply migrations from version B, shutdown A
- A/B testing: Version B is released to a subset of users under specific conditions while a similar group remains on version A
- Shadow: Version B receives real-world traffic alongside version A and doesn't impact the response

Image by Tumisu from Pixabay

3... 2... 1... Deployed

Image by Nora from Pixabay

Run acceptance tests

100%

GREEN

Release is going ↗ / Joy ↗

Image by Mohamed Hassan from Pixabay

Everything was happily tested
Is that's all?

Monitoring

Image by 巻（Maki） from Pixabay

Monitoring

- Logs
- Traces
- Metrics

Logs

Image by Clker-Free-Vector-Images from Pixabay

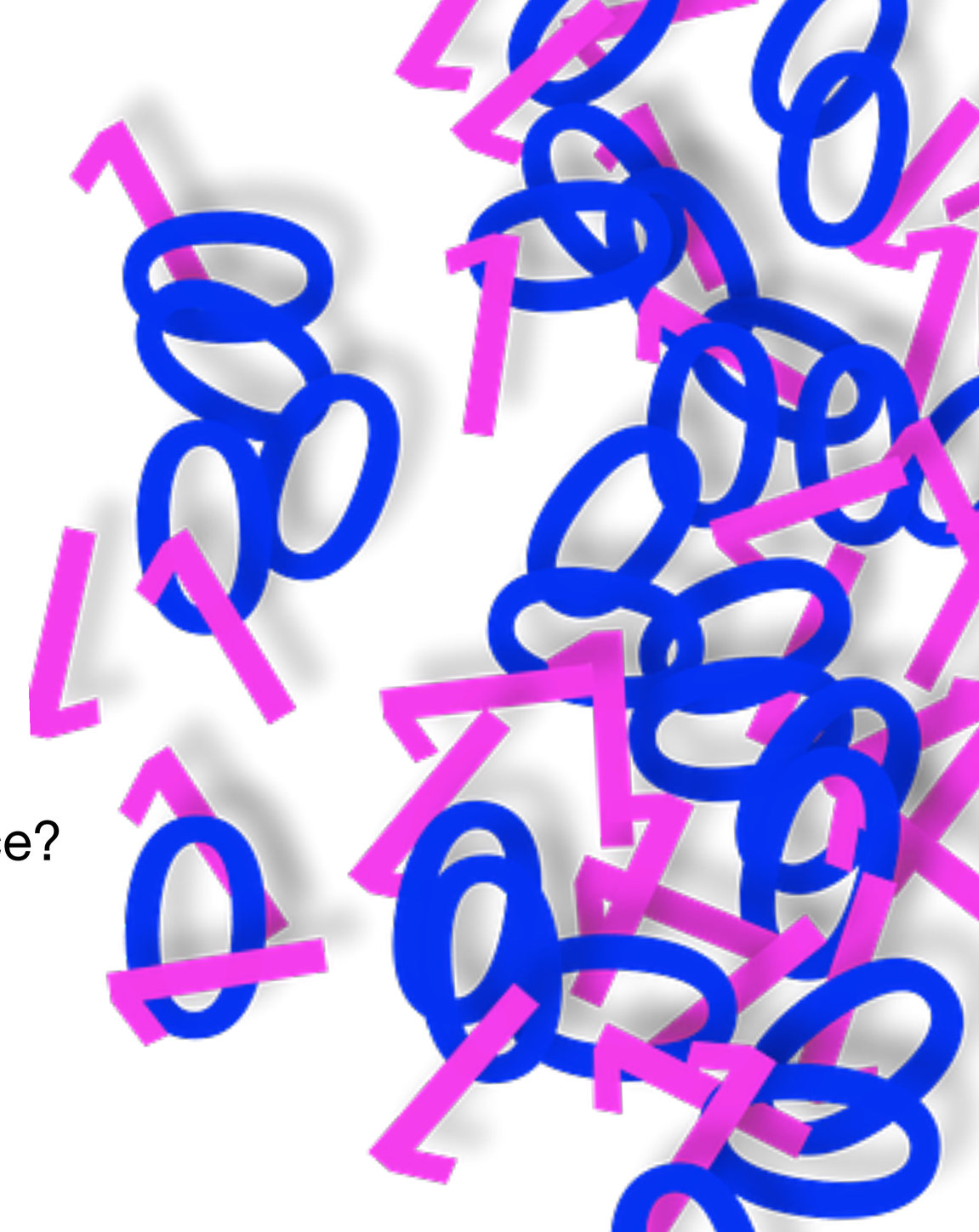
Logs

Logs are timestamped records of discrete events that occur within a system

In Python ecosystem logging usage is a mess

Logs

- Mess in setup behavior
- Or remove logging
- Decrease logging level for some libraries
- Questions why something wasn’t logged?
- Questions why was something logged twice?



Logs

Debugging handlers

```
<RootLogger root (DEBUG)>
     <TimedRotatingFileHandler /path/to/myapp.log (DEBUG)>
     <StreamHandler <stdout> (DEBUG)>
+ [concurrent.futures  ] <Logger concurrent.futures (DEBUG)>
+ [concurrent          ] <logging.PlaceHolder object at 0x7f72f624eba8>
+ [asyncio             ] <Logger asyncio (DEBUG)>
+ [myapp               ] <Logger myapp (DEBUG)>
+ [flask.app           ] <Logger flask.app (DEBUG)>
+ [flask               ] <Logger flask (DEBUG)>
+ [werkzeug            ] <Logger werkzeug (ERROR)>
```

stackoverflow.com/a/60988312

Logs

Debugging handlers

```
import logging

def listloggers():
    rootlogger = logging.getLogger()
    print(rootlogger)
    for h in rootlogger.handlers:
        print('     %s' % h)

    for nm, lgr in logging.Logger.manager.loggerDict.items():
        print('+ [%-20s] %s ' % (nm, lgr))
        if not isinstance(lgr, logging.PlaceHolder):
            for h in lgr.handlers:
                print('     %s' % h)
```

stackoverflow.com/a/60988312

Structured logs for searching in fields

Logs

Output in JSON or structlog

```
from pythonjsonlogger import jsonlogger
...
json_log.info({"message": "hello, world", "key": “value!"})
{"message": "hello, world", "levelname": "INFO","asctime": "2024-11-23
10:41:29,202", "key": “value!"}

import structlog
...
struct_log.info("hello, %s!", "world", key="value!")
2024-11-23 10:41:29 [info     ] hello, world!   key=value!
```

Traces

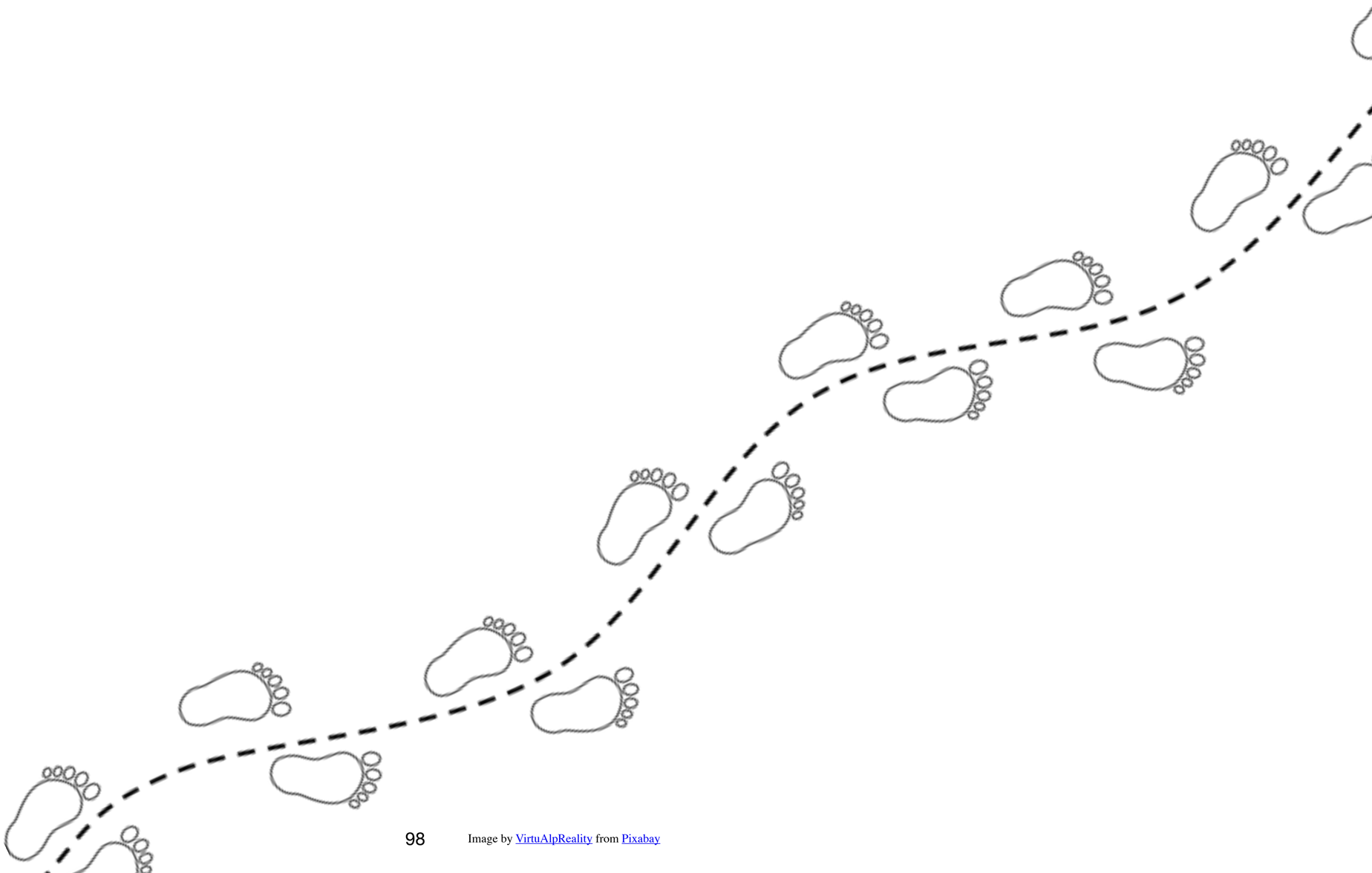


Image by VirtuAlpReality from Pixabay

Traces

Definition

Trace refers to the event records within a single request flow through a distributed system

Image by OpenClipart-Vectors from Pixabay

Image Mirosław i Joanna Bucholc from Pixabay

Traces

Custom span

```
from opentelemetry import trace
tracer = trace.get_tracer(__name__)

# ...
with tracer.start_as_current_span('signup.submit.push_applications'):
    await queue.push_applications_message_to_queue(
        queue_application_message,
        EventTypes.created,
    )
```

Traces

Library ready instrumentators

```
from prometheus_fastapi_instrumentator import Instrumentator

Instrumentator().instrument(app)
```

Traces

Library ready instrumentators

```
# from opentelemetry.instrumentation.(...) import
URLLib3Instrumentor, ...

URLLib3Instrumentor().instrument()
SQLAlchemyInstrumentor().instrument(
    enable_commenter=True,
    commenter_options={},
)
HTTPXClientInstrumentor().instrument()
AsyncPGInstrumentor().instrument()
```

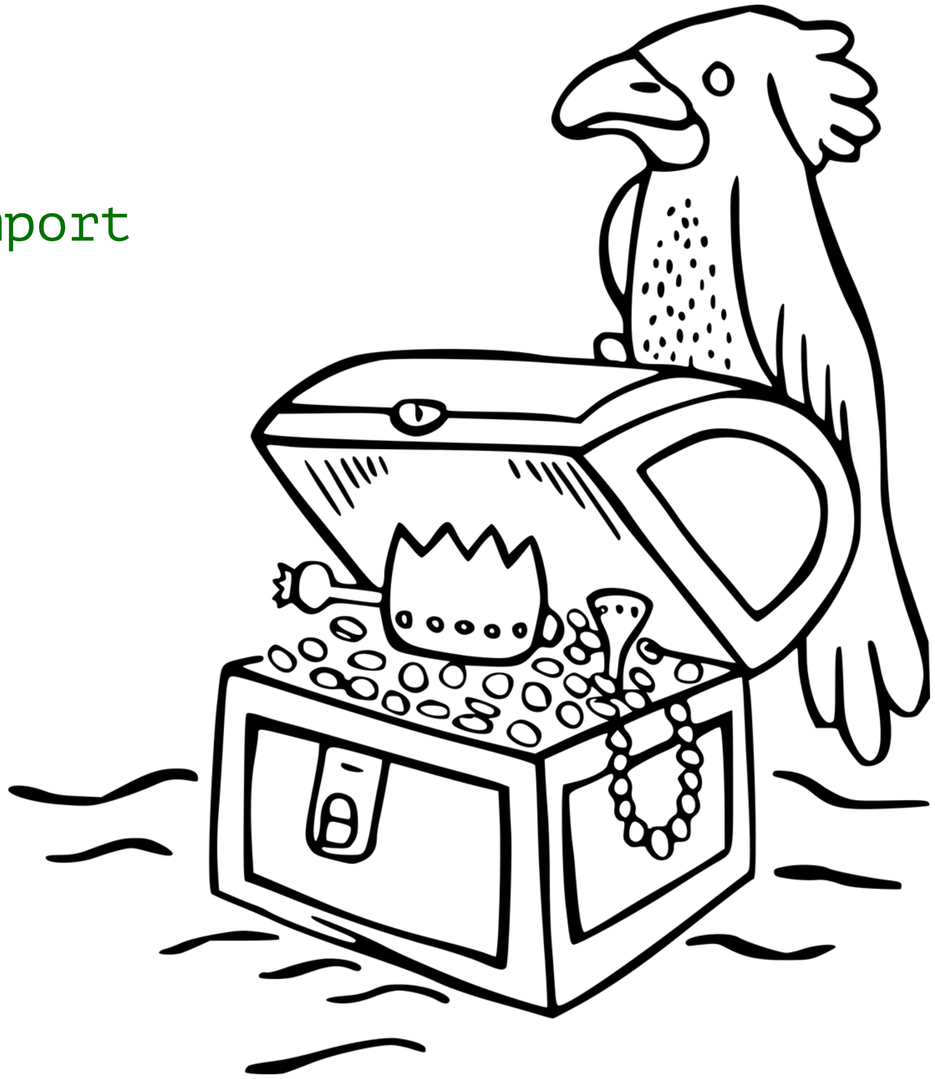


Image by Please support me! Thank you! from Pixabay

Metrics

Image by Yvette W from Pixabay

Metrics

Metrics are numerical data points that measure the performance or state of a system over time

Metrics

Custom middleware and metrics

```
from starlette.middleware.base import BaseHTTPMiddleware,
RequestResponseEndpoint

REQUESTS_PROCESSING_TIME = Histogram(
    'fastapi_requests_duration_seconds',
    'Histogram of requests processing time by path (in seconds)',
    ['method', 'path', 'status_code', 'app_name'],
)

class PrometheusMiddleware(BaseHTTPMiddleware):
    ...
```

Metrics

Custom middleware and metrics

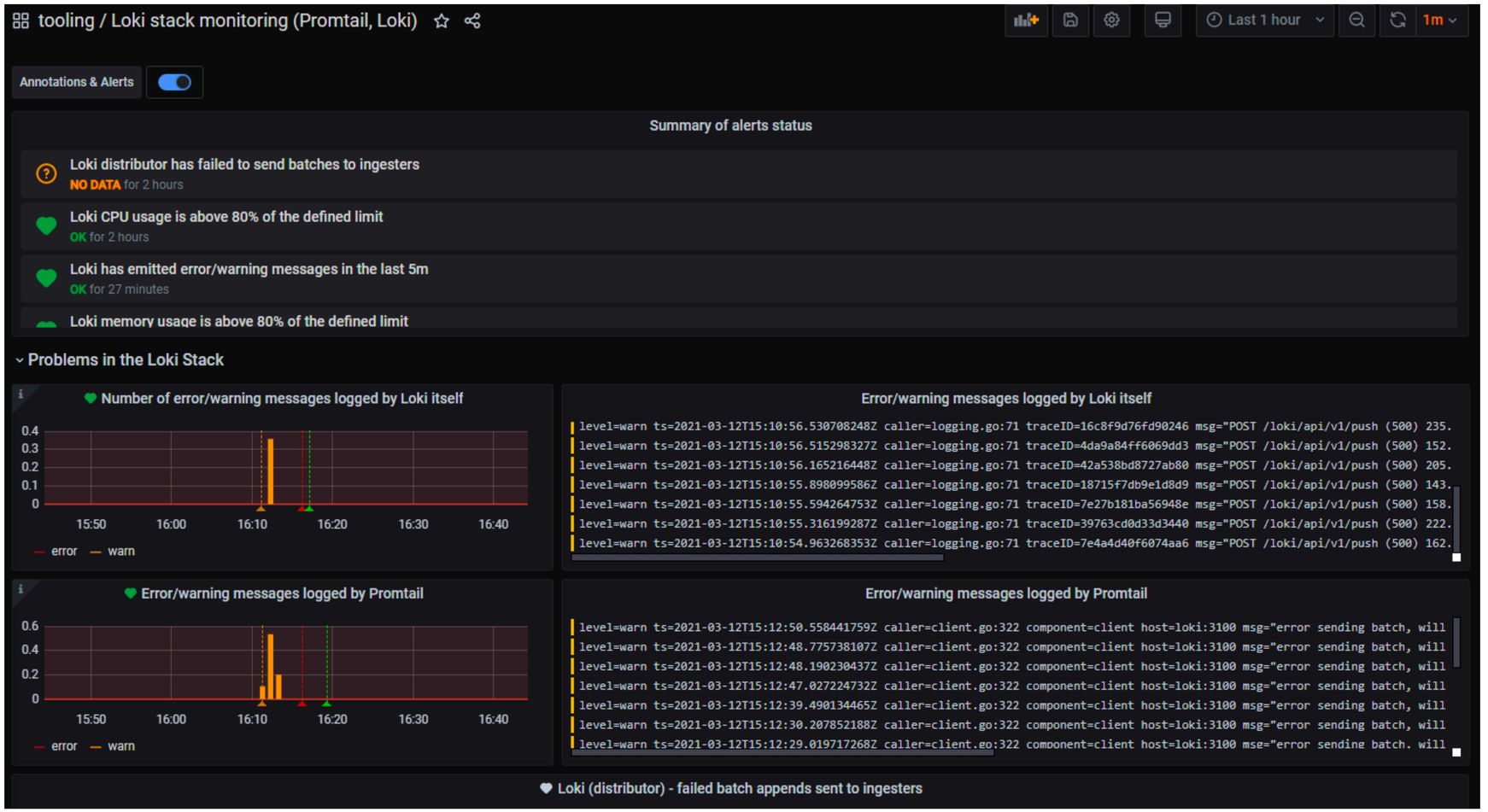
```
async def dispatch(
    self, request: Request, call_next: RequestResponseEndpoint
) -> Response:
    # ignore technical requests
    before_time = time.monotonic()
    status_code = 500
    try:
        response = await call_next(request)
        status_code = response.status_code
    finally:
        after_time = time.monotonic()
        REQUESTS_PROCESSING_TIME.labels(
            method=method, path=path, status_code=status_code, app_name=self.app_name
        ).observe(after_time - before_time)

    return response
```

Grafana

grafana.com/docs/grafana/latest/fundamentals/dashboards-overview/

Grafana



grafana.com/grafana/dashboards/14055-loki-stack-monitoring-promtail-loki/

Observability ↗ / Joy ↗

Image by Gordon Johnson from Pixabay

Does it look like this is it?

Yes!

- FastAPI can be enterprise-ready
- Not out of the box. Batteries need to be connected

Image by Mohamed Hassan from Pixabay

That’s it

Thank you!

linkedin.com/in/aptakhin

Alexander Ptakhin

Lead @ Prestatech GmbH

bsky: @aptakhin.name

mstdn: hachyderm.io/@AlexPtakhin

linkedin.com/in/aptakhin

github.com/aptakhin

aptakhin.name

Images with URL reference or DALL-E

Presentation file