Flow

2024 Spring

Модель архитектуры предприятия на графе

Роман Цирульников

10 money

Содержание

Определение задачи

Масштаб системы

>20

лет промышленной эксплуатации

>680

прикладных сервисов онлайн

>400

технических систем

Наши основные продукты

Электронный кошелёк
и банковские карты

Приём платежей
за товары и услуги

Расчётное обслуживание
и банковские карты
для юридических лиц

Эффекты масштаба и времени

- Постоянное накопление сложности систем
- **Неуправляемый рост сложности систем — один из основных рисков для новых проектов**
- Стоимость внесения изменений — это определяющий фактор развития

Основные задачи архитектуры

- **Фиксировать и распространять знания**
- **Ограничивать рост сложности систем и когнитивную нагрузку на команды**
- Преодолевать разрыв между предметной областью и технической командой
- Выстраивать матрицу ответственности за сервисы
- Обеспечивать предсказуемость и повторяемость результатов в последующих проектах
- Обеспечивать автономность команд

Имеем две задачи

#1: Спроектировать решение, как должно быть сделано

#2: Исследовать, что фактически есть сейчас

#1: Спроектировать решение

- Это аналитическая работа человека
- Человек работает над конкретным решением (продуктом, проектом, частью системы)
- Отвечает на вопрос «как должно быть **и почему**»

#2: Исследовать, что фактически есть сейчас

- Поиск ответов на очень разные вопросы
- Поиск по всей системе, по множеству решений
- Готовые представления не всегда подходят

О проектировании решений

- **Требует погружения в предметную область**
- Простой модели недостаточно, для описания предметной области нужен натуральный язык
- Подход **Docs as Code** отлично работает в этом классе задач
- Документы группируются по предметным областям

О документации as Code

- **Документы пишутся человеком в рамках задачи**
- Могут быть изменения систем, не отраженные в документах
- Удобно реализуется история изменений, коллективная работа, ревью
- Документация не является полноценной аналитической моделью
- Сложно сделать автоматизированный контроль систем

Об аналитических моделях

- **Модели формализованы, предоставляют конечный набор атрибутов**
- Удобно реализовывать автоматизированный контроль систем
- Сложно описать предметную область, не заменяет документацию
- Отражает факты о системе, но не объясняет причины
- Сложно реализовать историю изменений, особенно их визуализацию

Наш подход

Проектирование на основе документации as Code

Собирать **модель** из фактических конфигураций систем

Метамодель определяет **требования** к документам репозитория и объектам модели

Задачи модели

- Инструмент исследования системы при проектировании решений
- Инструмент аудита системы
- Инструмент анализа зависимостей техсистем, продуктов, услуг, команд
- Инструмент сверки документации с действительностью

Метамодель

В основе метамодели

- TOGAF
- ArchiMate
- ISO-20000 IT Service Management (ITSM)
- ITIL Foundation
- BIZBoK

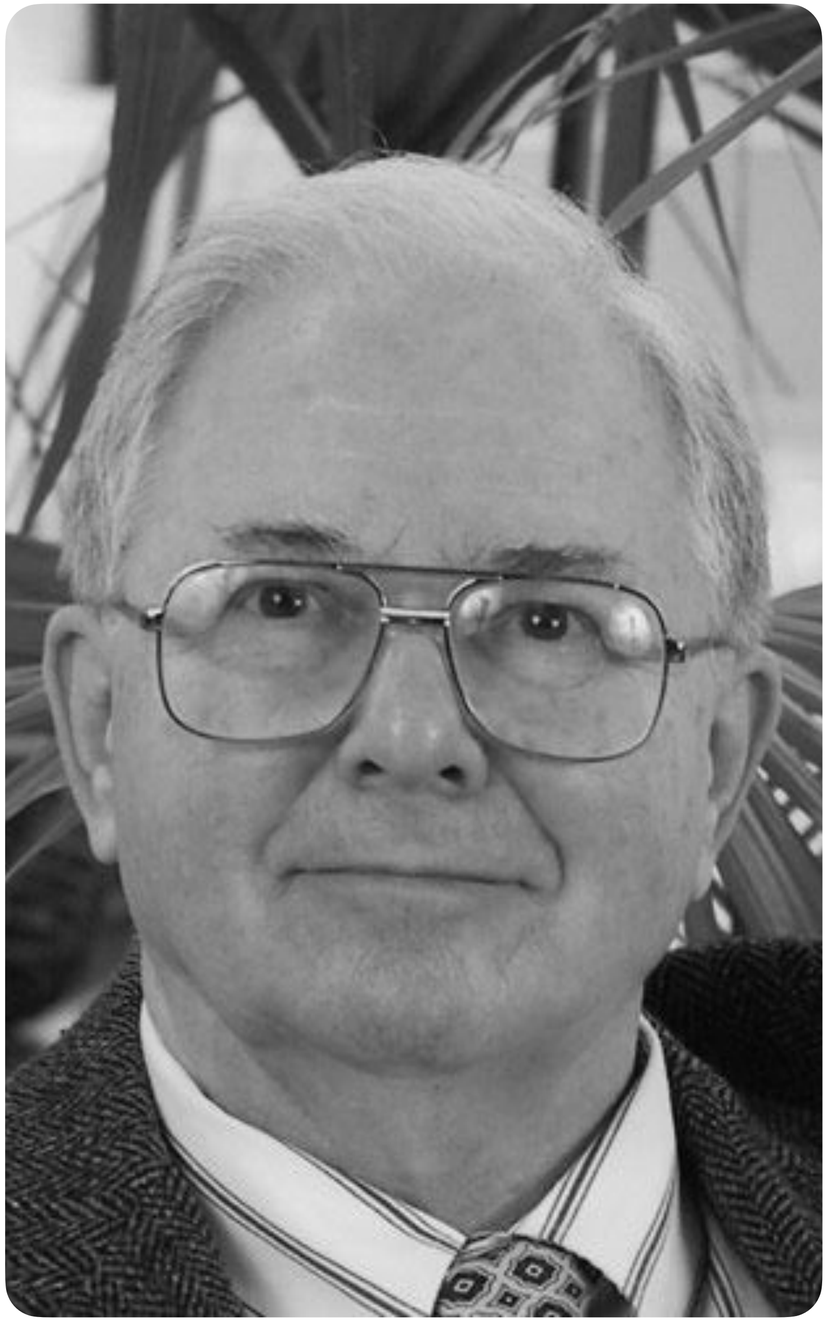
Принципы выбора

- **Международный стандарт** в основе, не выдумывать велосипедов
  - есть эксперты на рынке
  - совместимость с инструментами
- Выбрать **минимально достаточный** для нас набор сущностей
  - ограничиваем когнитивную сложность
  - трудоёмкость сопровождения как основное ограничение

Gall’s law

- Сложная система, созданная с нуля, никогда не заработает. Начинать надо с работающей простой системы.

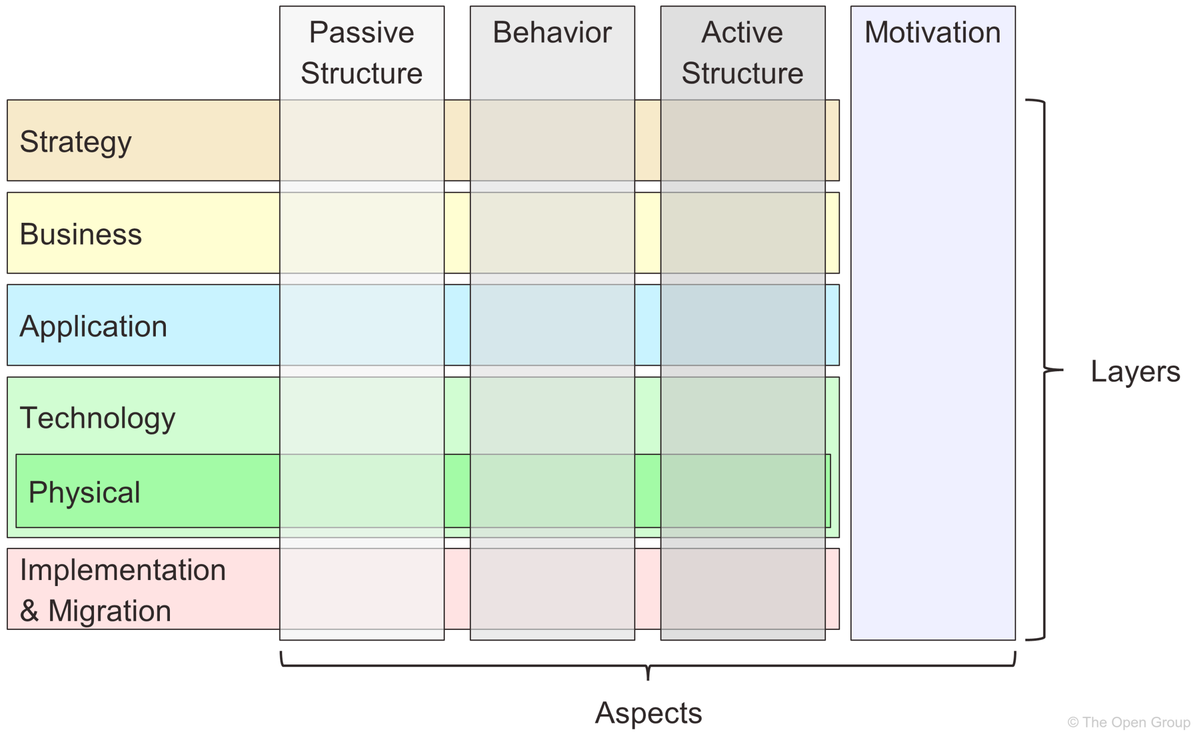
John Gall, Systemantics: How Systems Really Work and How They Fail, 1977



В основе декомпозиции

1. **Сервис** (Услуга) — единица видимого поведения
2. Продукт как агрегация набора услуг
3. Нотация ArchiMate
4. Слой модели отражает ответственность (департамент)
    - Эксплуатация
    - Разработка
    - Развитие бизнеса

Слои ArchiMate



- **Technology** — сервисы инфраструктуры
- **Applications** — сервисы прикладных решений
- **Business** — продукты и услуги
- **Strategy+Motivation** — ключевые компетенции и сегменты потребителей

Объекты слоя Technology

- Отражают область ответственности подразделений эксплуатации
- **Technology Service** — сервис инфраструктуры, например S3
- **Technology Interface** — интерфейс подключения, например api.yookassa.ru
- **Technology Component** (System Software) — ПО инфраструктуры, например mail relay, СУБД
- **Technology Path** — канал коммуникации систем, например топик Kafka
- **Technology Node** — кластер оборудования
- **Technology Artifact** — база данных (данные)

Объекты слоя Applications

- Отражают область ответственности команд разработки
- **Application Service** — прикладной сервис, реализуемый приложениями
- **Application Component** — приложение, единица развертывания в облако

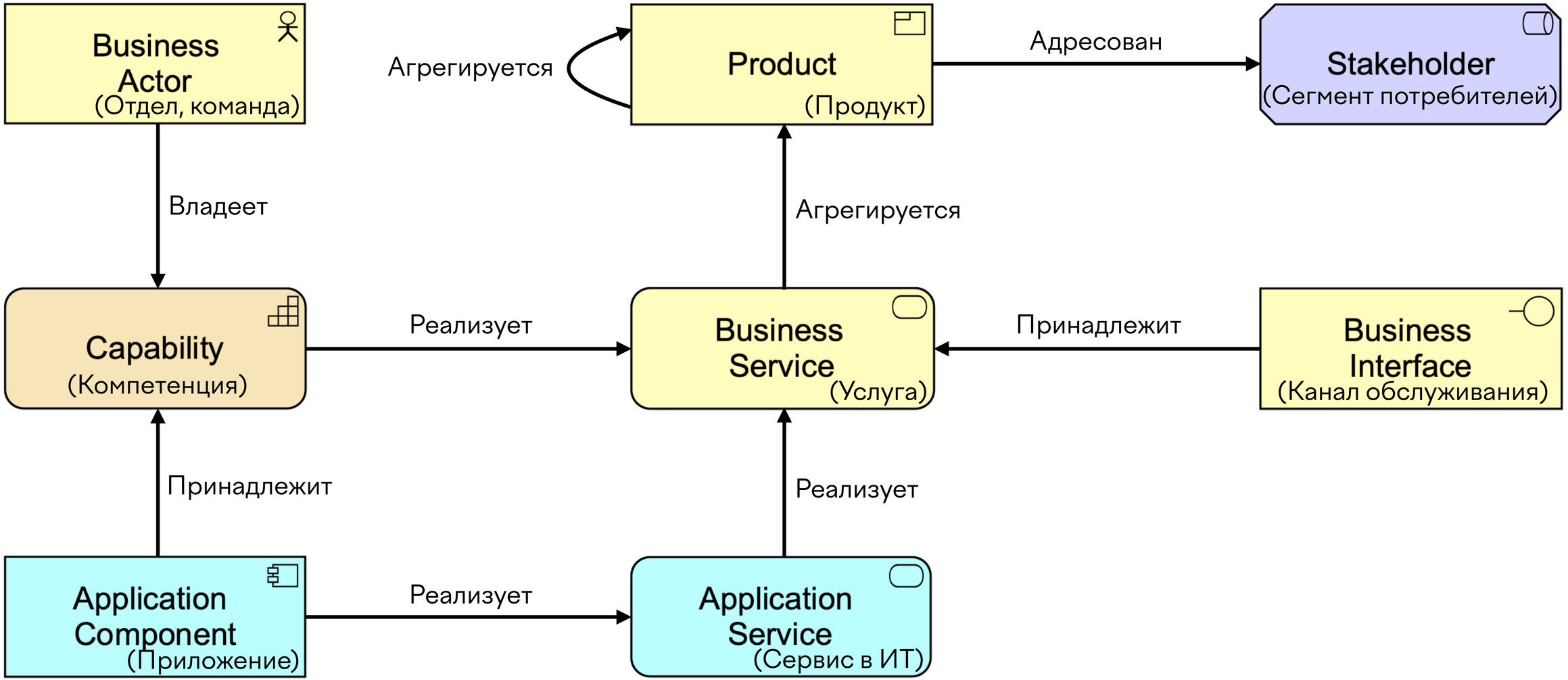
Слой Business

- Отражает **структуру продуктов и услуг** компании
- **Продукт** не существует в отрыве от:
  - сегмента потребителей
  - компетенций компании (capabilities)
- ArchiMate не предоставляет внятной метамодели связей Business, Strategy, Motivation
- Приняли решение схлопнуть объекты Business, Strategy, Motivation в общий слой

Объекты слоя Business

- **Business Service** — услуга, предоставляемая конечным потребителям
- **Business Interface** — канал предоставления услуги (веб-сайт, API, контакт-центр)
- **Product** — агрегация набора услуг для реализации ценности определенной группе потребителей
- **Stakeholder** — сегмент потребителей (шоперы, геймеры, интеграторы)
- **Capability** — компетенция в определенной предметной области (эквайринг банковских карт)
- **Business Actor** — подразделение компании (отдел, команда, служба)

Метамодель слоя Business



Реализация

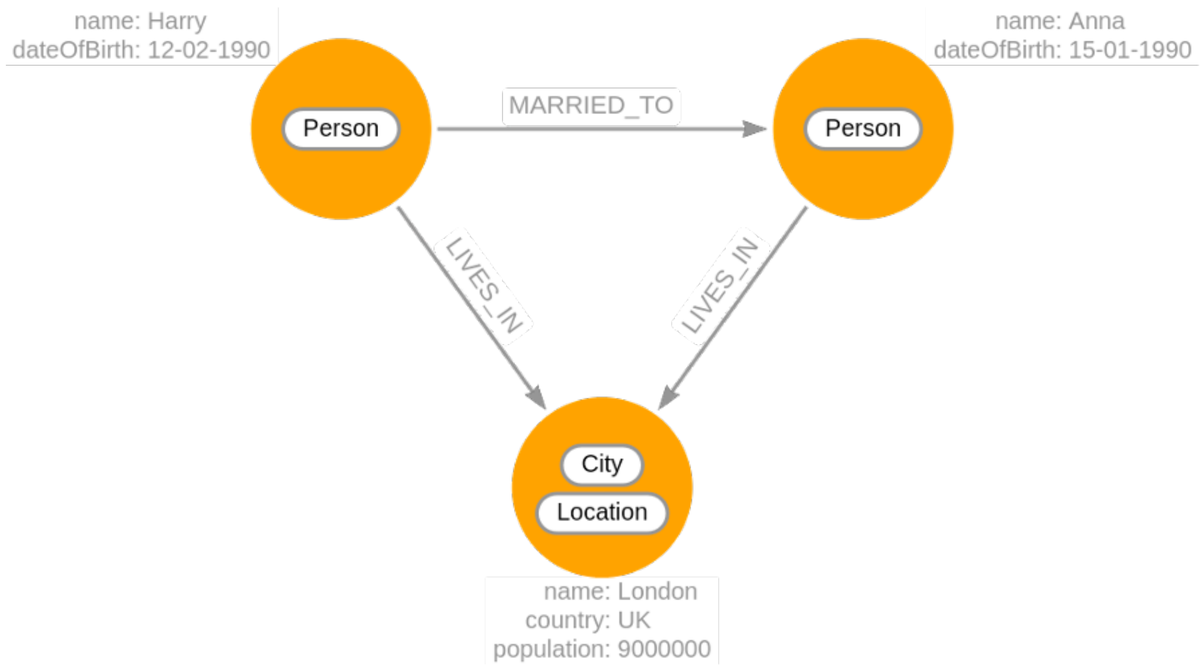
Если пытаемся построить общую схему

- Рост количества элементов делает схему нечитаемой
- С определенного уровня сложности схему становится невозможно отразить на плоскости
- Architecture viewpoints: разным интересантам нужны ответы на разные вопросы

Структура данных граф (graph)

- Может содержать в себе большое количество элементов
- Позволяет производить частичные выборки по разным условиям
- Позволяет визуализировать выбранные данные

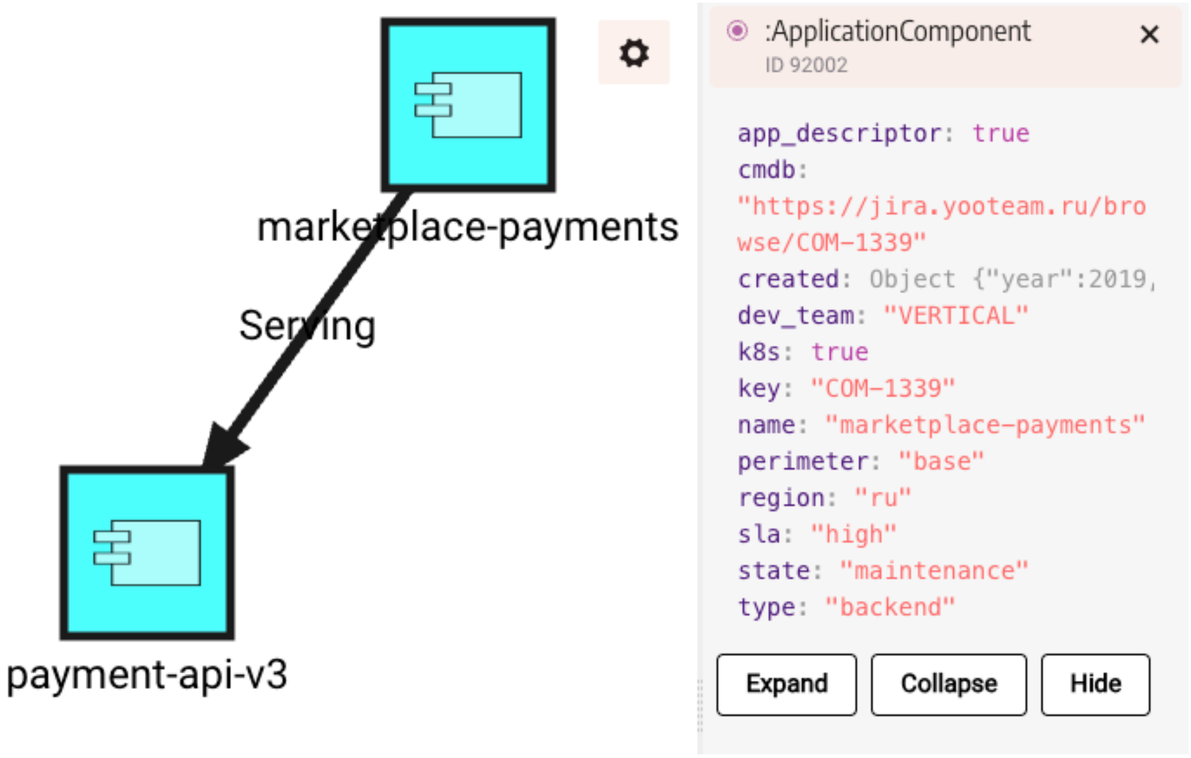
Основные понятия



- **Node** (Узел) — отражает сущность
- **Relation** (Отношение aka Ребро) — отражает взаимосвязь двух Node
- **Label** определяет тип Node или Relation
- **Properties** определяет свойства Node или Relation

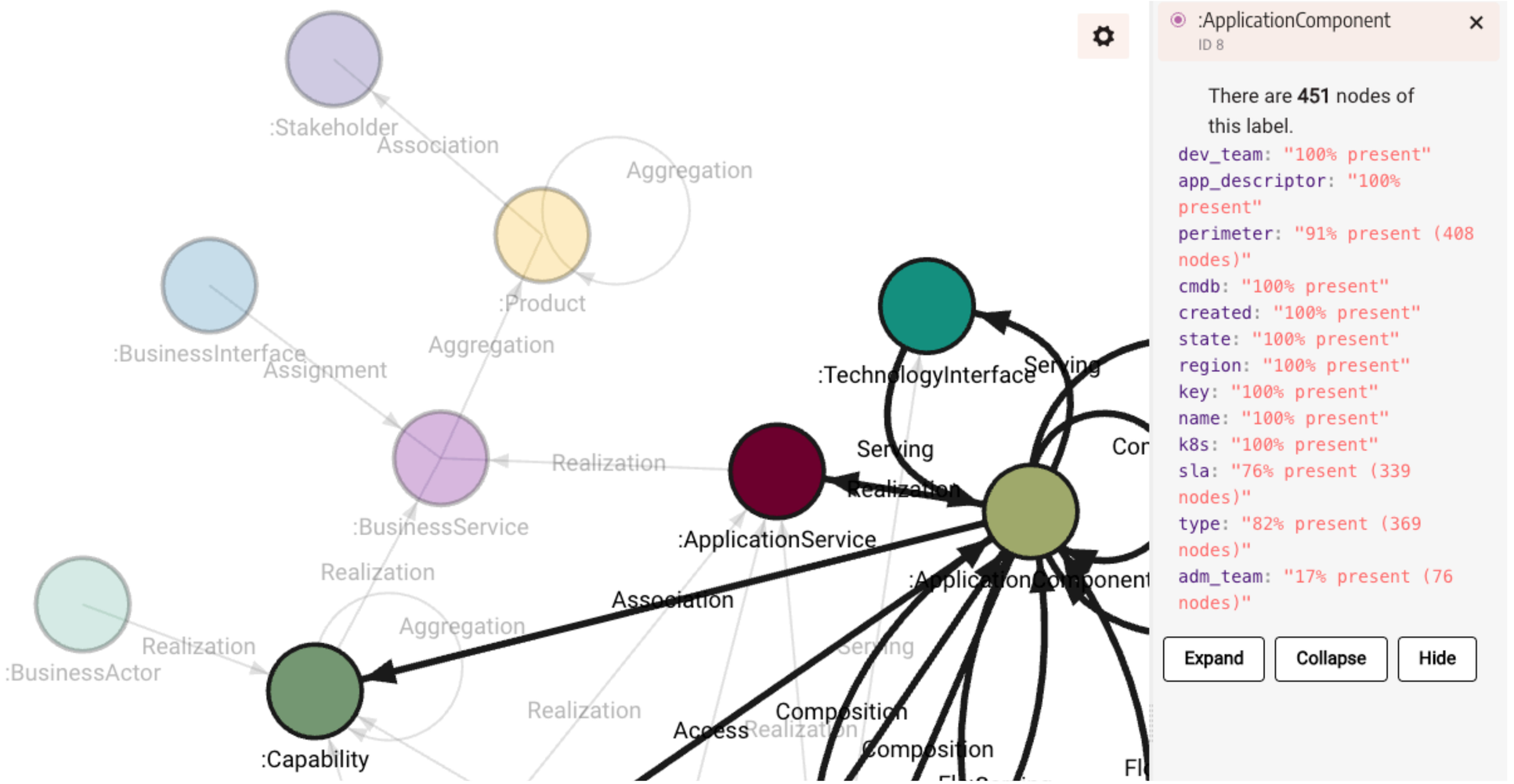
https://memgraph.com/docs/querying/create-graph-objects

Типы данных



- Узлы и отношения графа строго типизированы
- Можно делать аналитические запросы
- Можно компоновать условия
- Можно трассировать пути

Схема графа


:Stakeholder
Association
Aggregation
:Product
:BusinessInterface
Assignment
Aggregation
:BusinessService
Realization
:TechnologyInterface
Serving
Serving
Realization
:ApplicationService
:ApplicationComponent
Association
Realization
Aggregation
:BusinessActor
Realization
:Capability
Realization
Access
Composition
Composition
Serving
:ApplicationComponent
ID 8
There are 451 nodes of this label.
dev_team: "100% present"
app_descriptor: "100% present"
perimeter: "91% present (408 nodes)"
cmdb: "100% present"
created: "100% present"
state: "100% present"
region: "100% present"
key: "100% present"
name: "100% present"
k8s: "100% present"
sla: "76% present (339 nodes)"
type: "82% present (369 nodes)"
adm_team: "17% present (76 nodes)"
Expand
Collapse
Hide

Графовые СУБД

- Neo4J
- Memgraph

Источники данных

- CMDB (учёт сервисов, систем, объектов инфраструктуры)
- Конфигурации объектов облака
- Конфигурации as Code в BitBucket
- Staff (сотрудники, команды, отделы, предметные области)
- Статический справочник сущностей в BitBucket

Технологическая зрелость

- Подход CloudNative формализует объекты инфраструктуры
- Приложения выполняются в стандартизованных конфигурациях
- Подход as Code формализует сущности, конфигурации
- Данные для модели можно автоматизированно собирать скриптами

Примеры

Язык запросов openCypher

- Это адаптация SQL для графов
- Базовый синтаксис:

```
MATCH (node1)-[rel]-(node2) RETURN node1,rel,node2;
```

Язык запросов openCypher

- Добавляете необходимые атрибуты и типы:

```
MATCH (n:ApplicationComponent {name: "appname"})-[r]-(m:TechnologyInterface)

RETURN n, r, m;
```

Язык запросов openCypher

- Если нужно логически сочетать несколько условий отбора:

```
MATCH (n:ApplicationComponent {name: "appname"})-[r]-(m)

WHERE m:ApplicationComponent OR m:TechnologyInterface

RETURN n, r, m;
```

Язык запросов openCypher

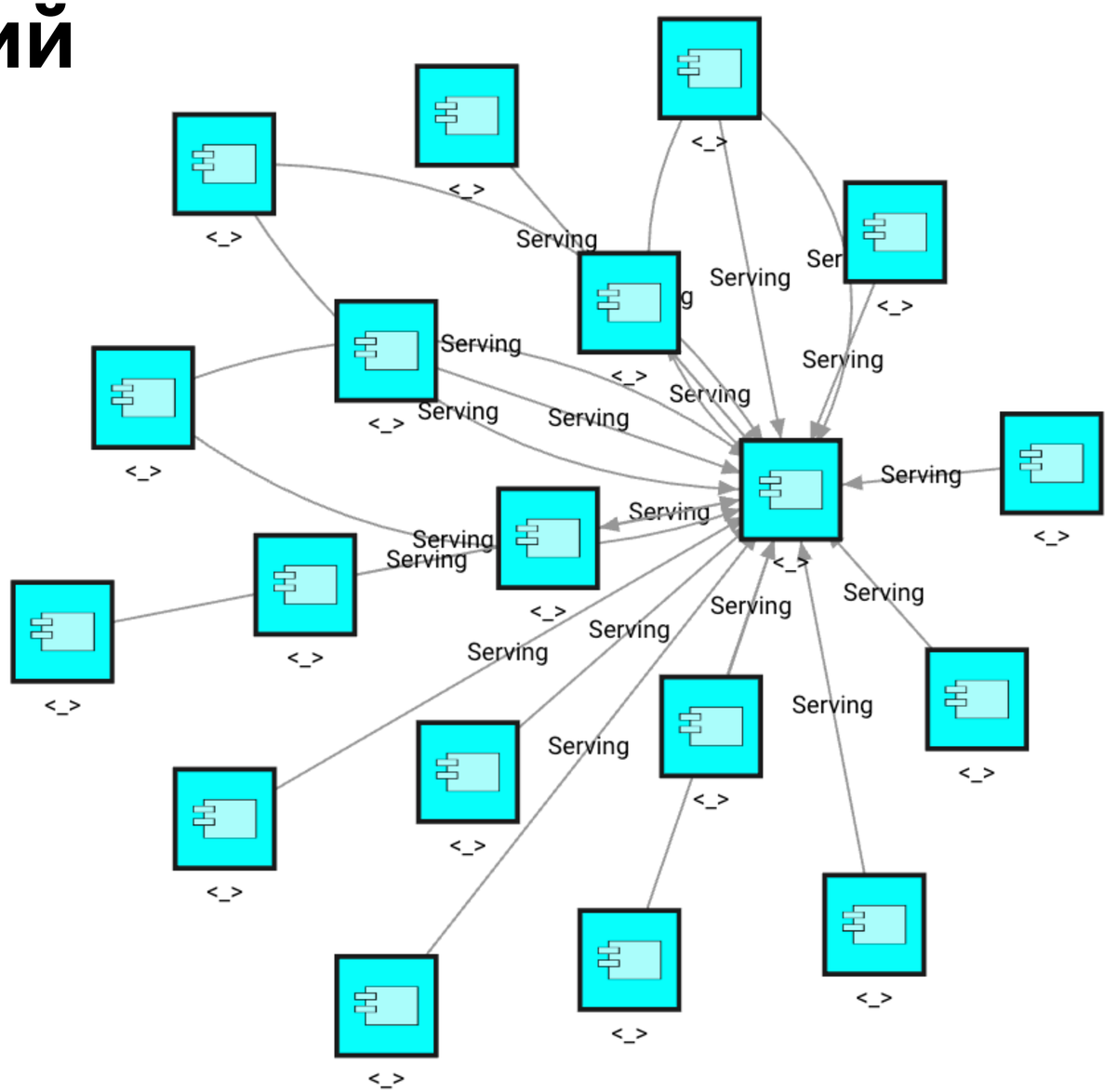
- Можно определить условие на направление отношения:

```
MATCH (n:ApplicationComponent {name:"appname"})<-[r]-(m) RETURN n, r, m;
```

Зависимости приложений

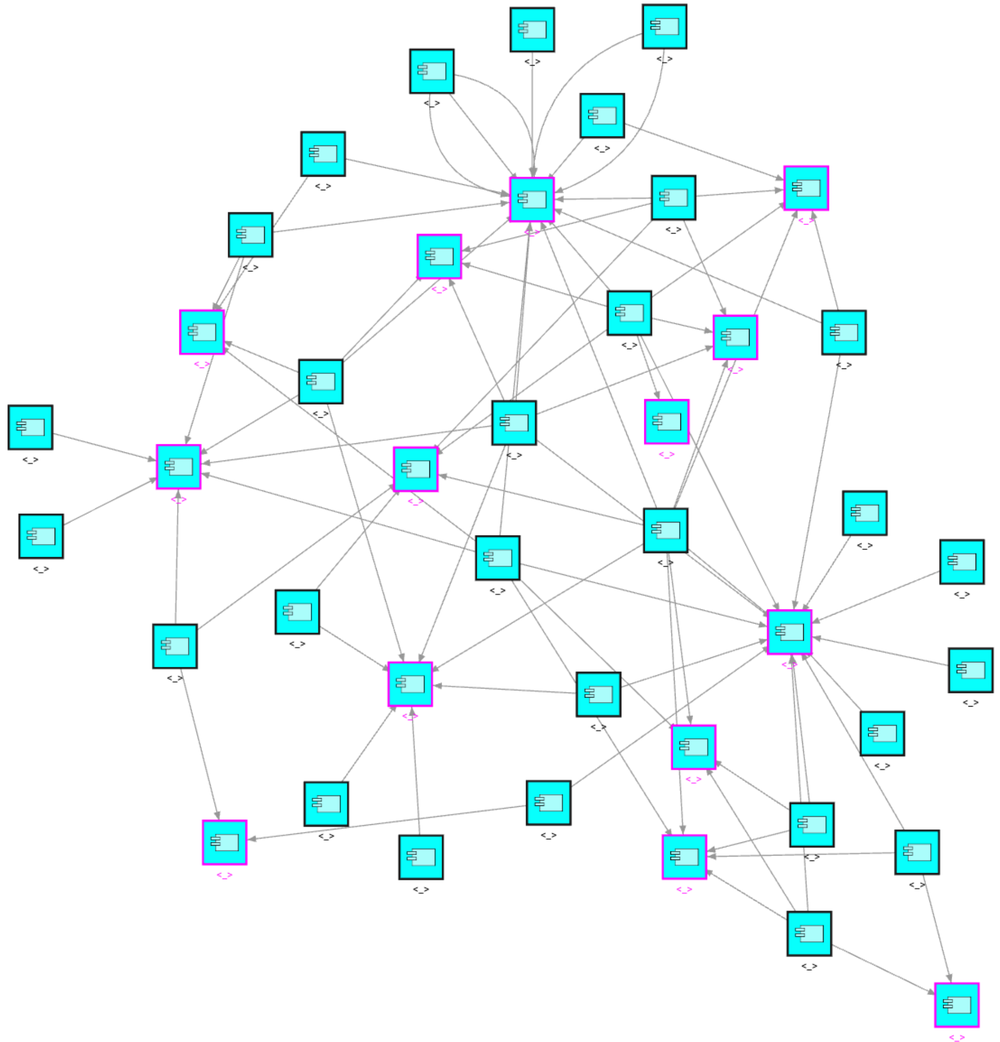
```
MATCH (n:ApplicationComponent
 {name: "appname"})-[r]-
(m:ApplicationComponent)
RETURN n, r, m;


MATCH (n:ApplicationComponent
 {type: "frontend"})-[r]-
(m:ApplicationComponent
 {type: "frontend"})
RETURN n, r, m;
```



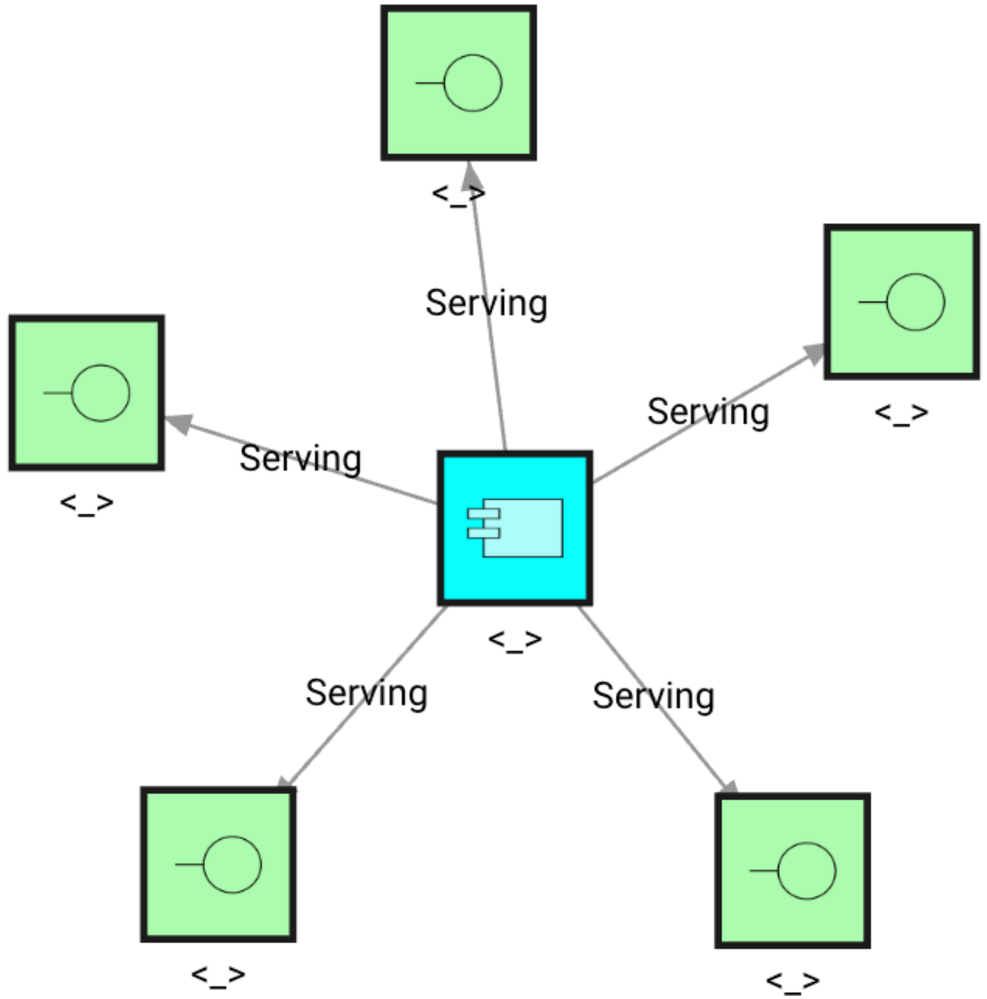
Контроль соблюдения периметров сетей

```
MATCH (n:ApplicationComponent
{perimeter: “pcidss"})
<-[r]-
(m:ApplicationComponent
{perimeter: "base"})
RETURN n, r, m;
```



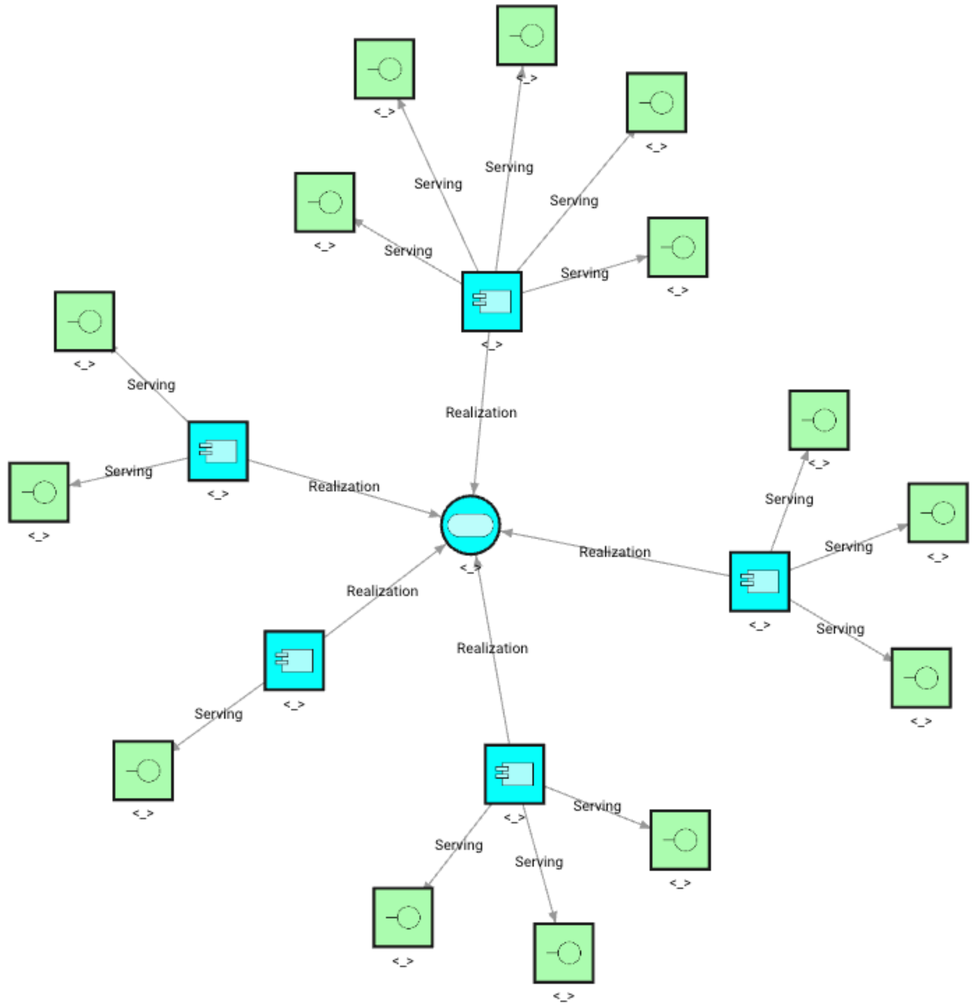
В какие адреса отражается приложение

```
MATCH (n:ApplicationComponent
{name: “appname"})
-[r]-(m:TechnologyInterface)
RETURN n, r, m;
```



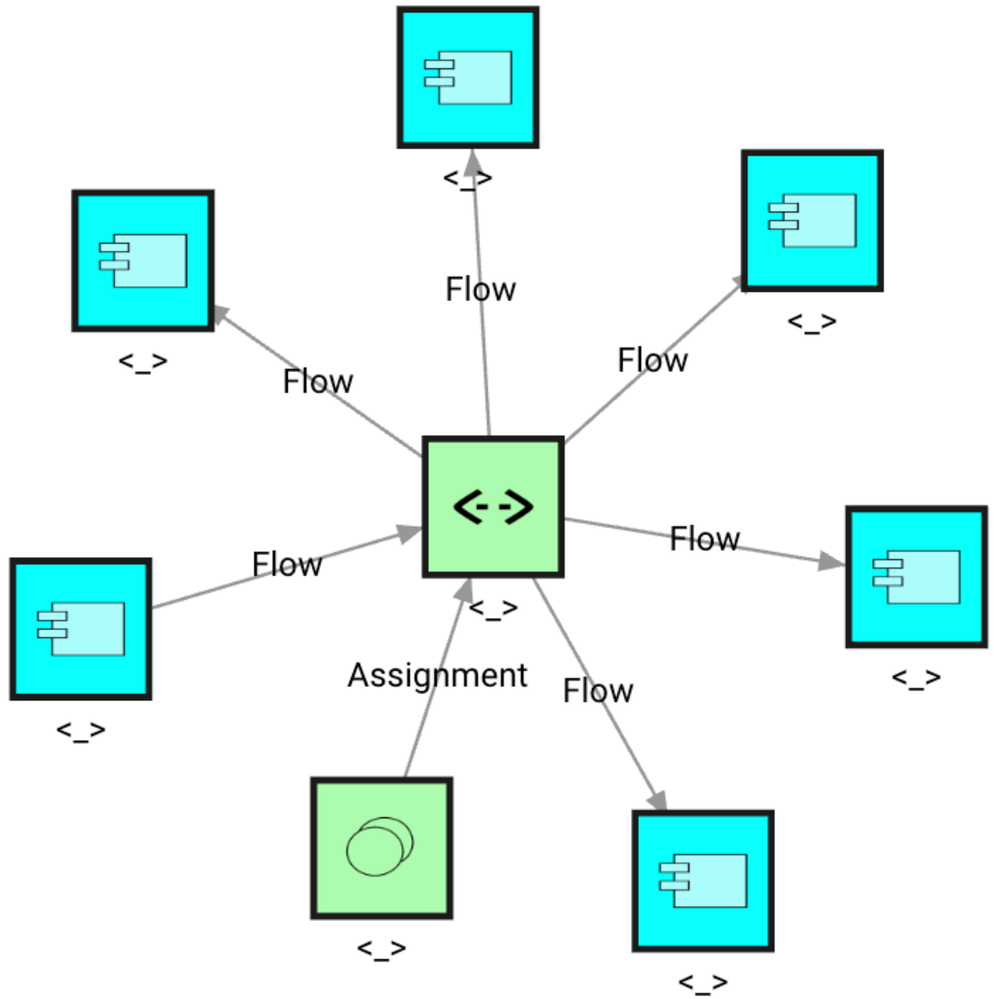
Из каких приложений и адресов состоит сервис

```
MATCH (n:ApplicationService
 {key: “SERV-1472”})
-[r]-(m:ApplicationComponent)
-[r2]->(m2:TechnologyInterface)
RETURN n, r, m, r2, m2;
```



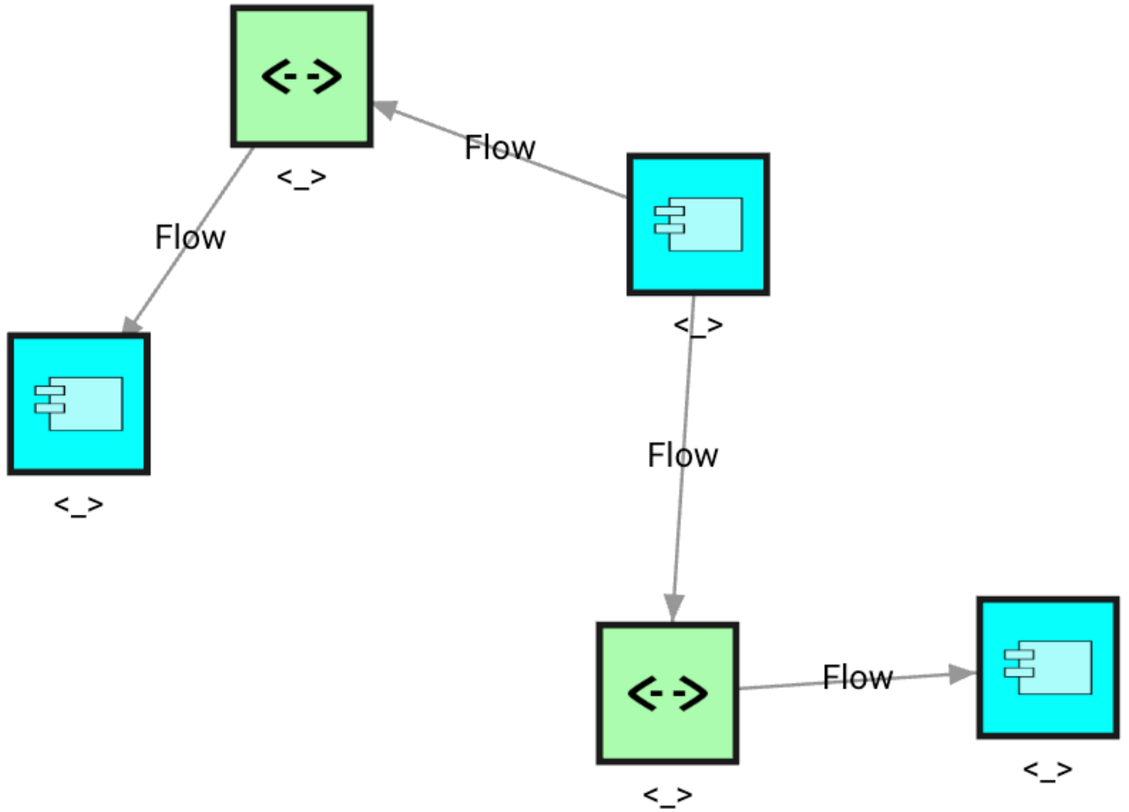
Потоки событий (данных MQ)

```
MATCH (n:TechnologyPath
{name:”kafka_topicname"})
-[r]-(m)
RETURN n, r, m;
```



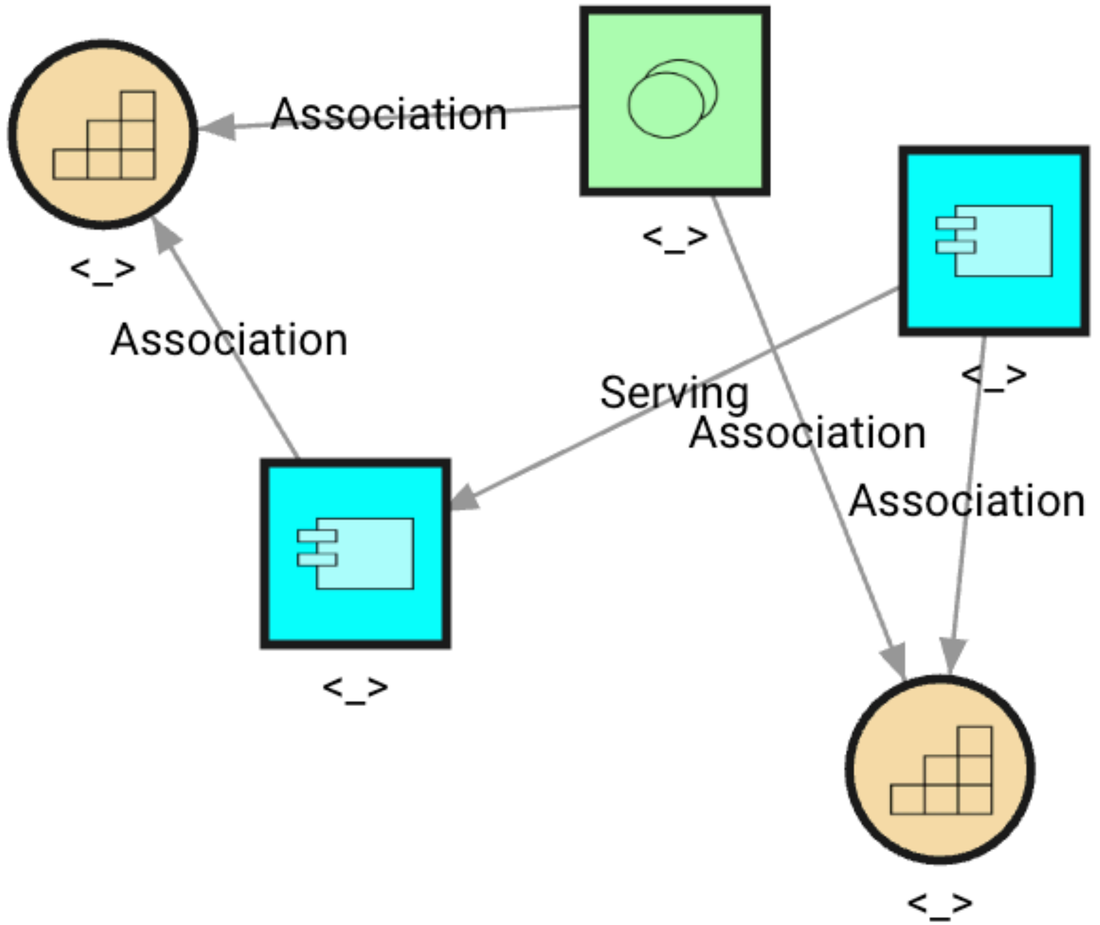
Показать потребителей данных BI систем

```
MATCH (s:ApplicationComponent
{type:”bi"})
-[r1]->(p:TechnologyPath)
-[r2]->(d:ApplicationComponent)
RETURN s,r1,p,r2,d;
```



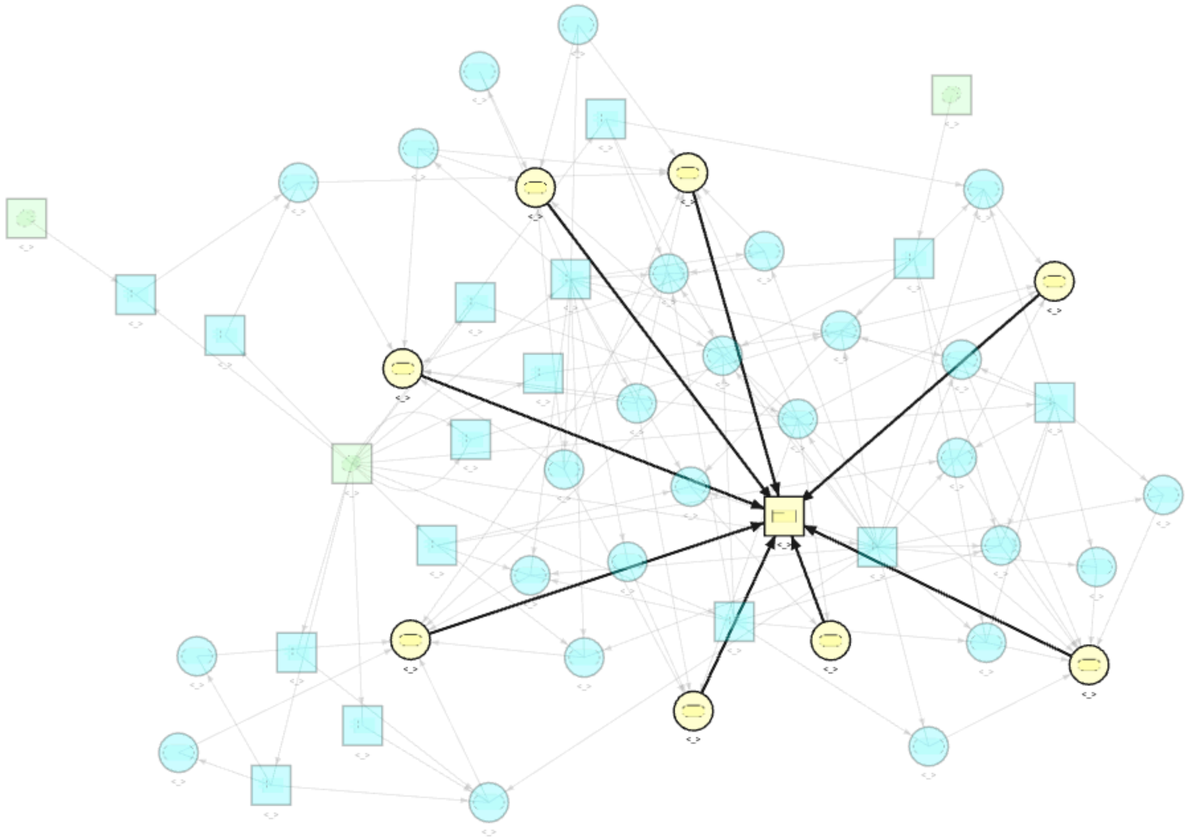
Пересечения предметных областей и команд

```
MATCH (n:Capability{key:”a**”})
MATCH (m:Capability{key:”i**”})
CALL algo.all_simple_paths(n, m,
[], 3)
YIELD path AS result RETURN
result;
```



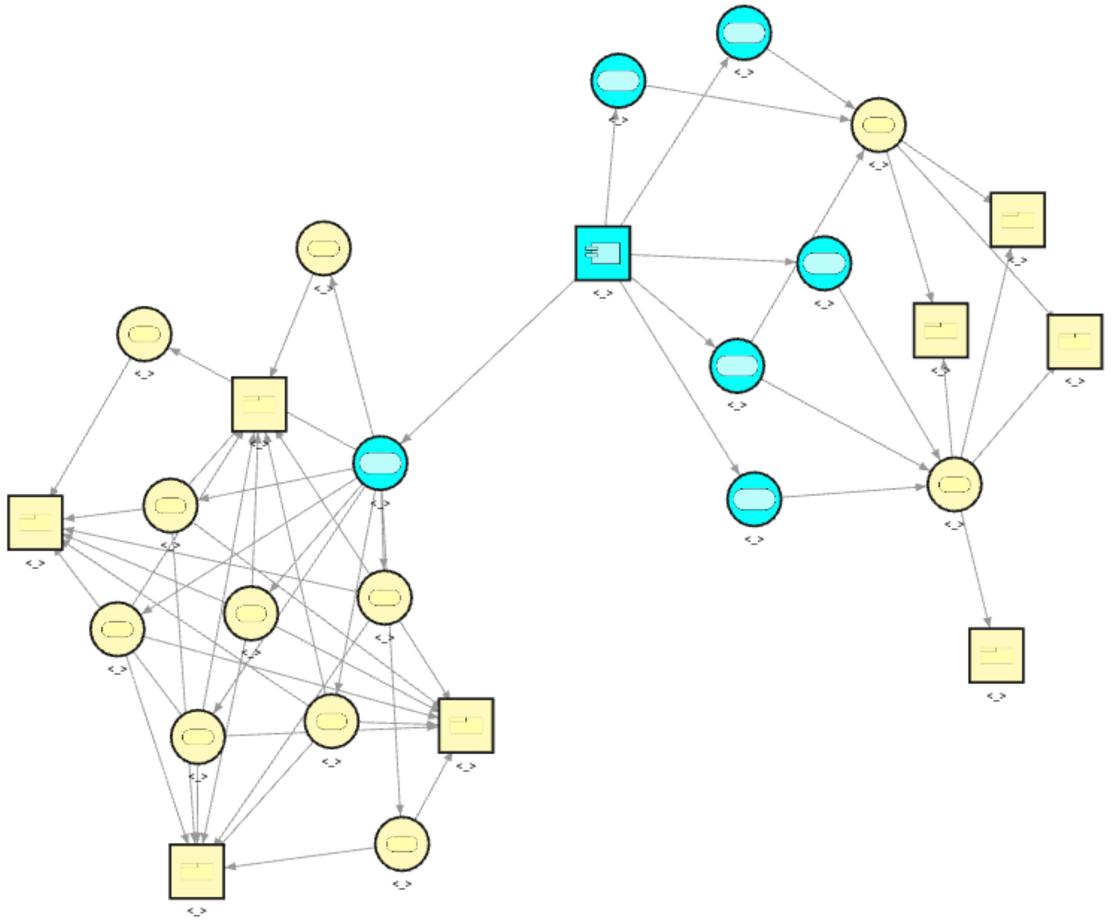
Из каких компонент состоит продукт

```
match(p:Product
{key:"productname"})-[r1]-
(bs:BusinessService)-[r2]-
(as:ApplicationService)-[r3]-
(ac:ApplicationComponent)-[r4]-
(tc:TechnologyComponent)
return p,r1,bs,r2,as,r3,ac,r4,tc;
```



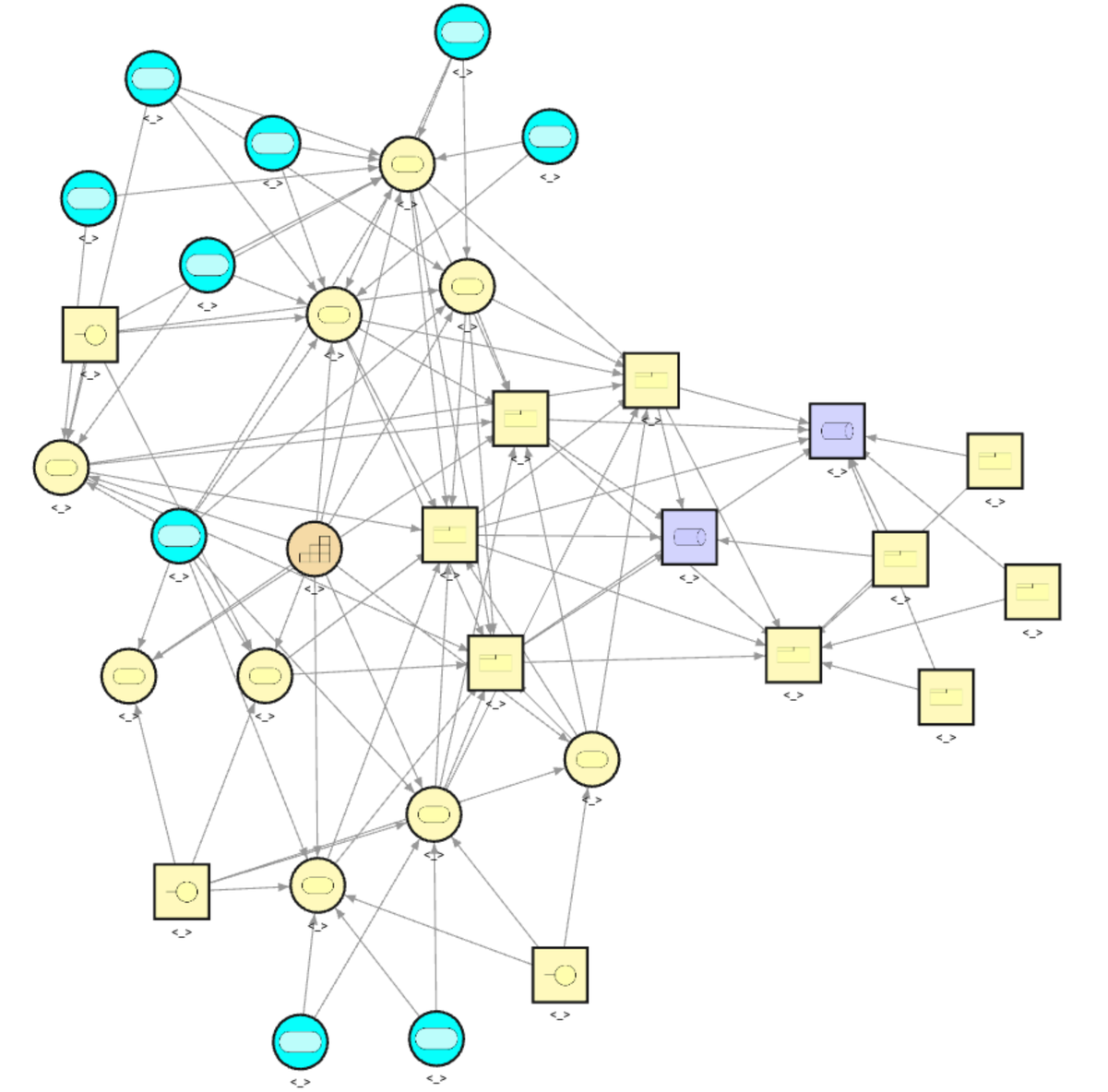
На какие продукты влияет приложение

```
match (ac:ApplicationComponent
{name:"appname"})-[r1]-
(as:ApplicationService)-[r2]-
(bs:BusinessService)-[r3]-
(p:Product)
return ac,r1,as,r2,bs,r3,p;
```



Зависимости продуктов

```
MATCH (n:Product
{key:”y__p__”})
MATCH (m:Product
{key:”y__c__”})
CALL algo.all_simple_paths(n, m,
[], 4)
YIELD path AS result RETURN
result;
```



В заключение

В заключение

- Серебряной пули нет (универсальных решений не бывает)
- Исследуйте бизнес-процессы и артефакты организации как источники данных модели
- Организации все разные, подбирайте подходящие инструменты и нотацию
- Ограничивайте сложность модели
- Облачные технологии значительно упрощают задачу

money

Благодарю за внимание

Роман Цирульников,
архитектор

romanvt@yoomoney.ru
@romanvt