

**avito.tech**

2023

# Скрывая Kubernetes: подходы к конфигурированию

**Лукьянченко Александр**

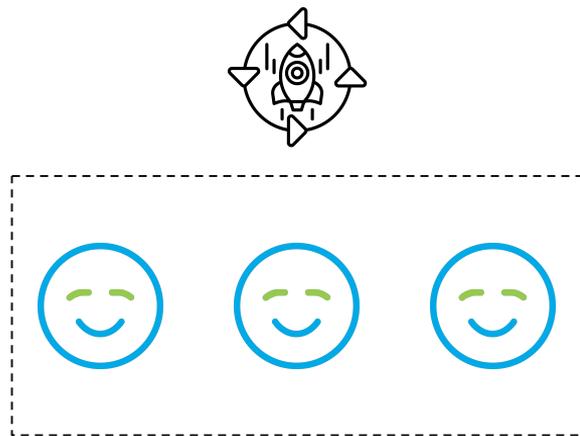
Руководитель разработки PaaS, Avito

# План

1. Платформа и уровень сервиса
2. Эволюция конфигурирования
3. Готовые реализации
4. Релизный цикл

# Платформа и уровень сервиса

# Devops

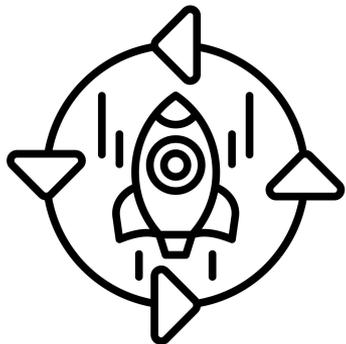


продуктовая команда

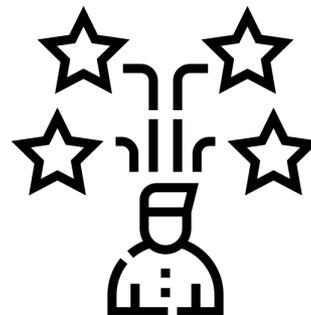


managed Kubernetes

# Devops

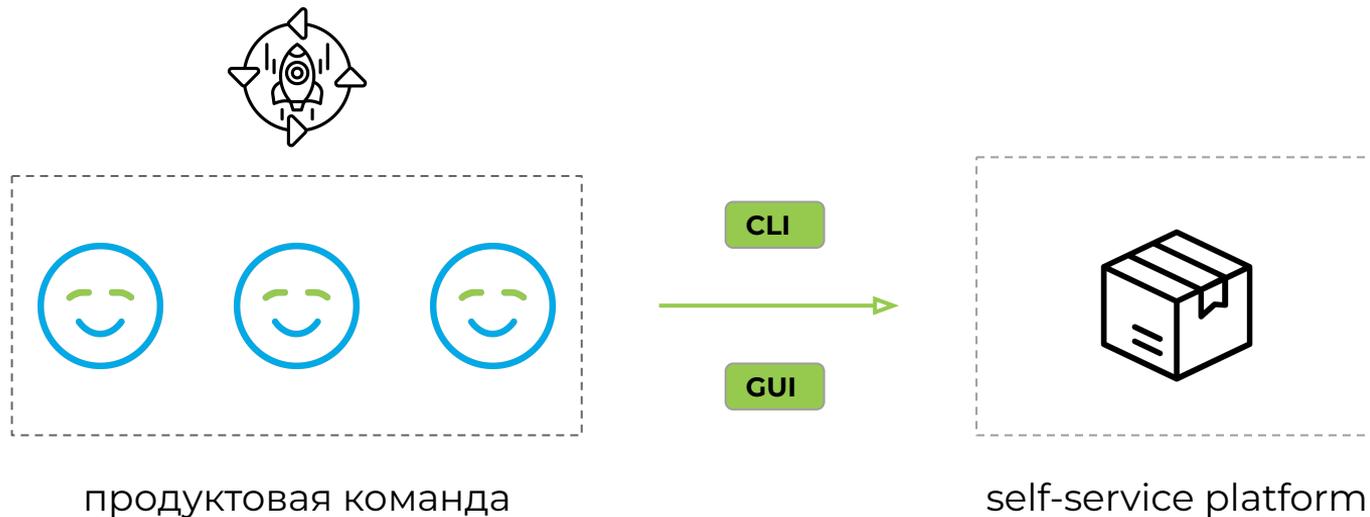


Владение всем  
жизненным циклом  
разработки



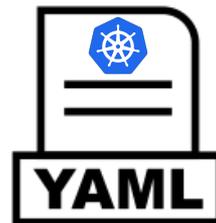
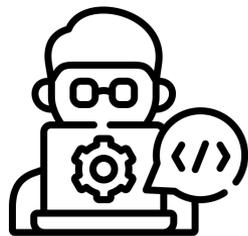
Обладание  
необходимыми навыками

# Централизованная платформа



# Эволюция конфигурирования

# Этап 1



# Этап 1: проблемы

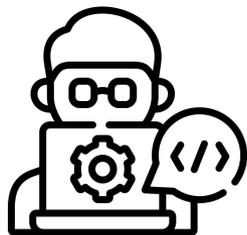
## **Высокий порог входа:**

1. Разработчик должен хорошо знать все примитивы куба
2. Правки порождают ошибки и инциденты в проде

## **Сложность конфигурирования**

3. Сложно управлять, множество разрозненных манифестов
4. Нужно учитывать окружения, иметь возможность точно менять значения
5. Мультикластер и сложная инфраструктура под капотом создаёт трудности унификации / поддержки конфигураций
6. Обновления k8s и инфра компонентов – боль

## Этап 2



# Этап 2: проблемы

## **Высокий порог входа:**

1. Разработчик должен хорошо знать все примитивы куба
2. Правки порождают ошибки и инциденты в проде

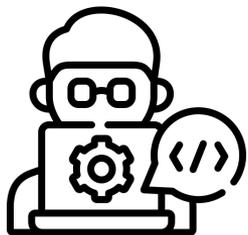
## **Сложность конфигурирования**

3. Обновления k8s и инфра компонентов – боль

# Что мы хотим?

1. Упростить работу с kubernetes, в идеале вообще не требовать знания k8s от разработчиков.
2. Предоставить все необходимые возможности по настройке и эффективному управлению приложениями, которые пишут разработчики.
3. Self-service механики на всех этапах жизненного цикла разработки.

# Этап 3



app.toml

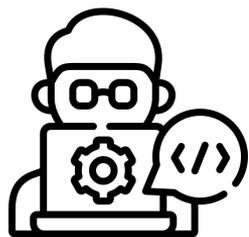
# app.toml

```
name = "user"  
description = "process user info"  
replicas = 1
```

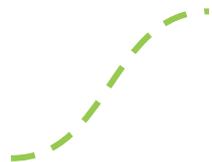
```
[engine]  
name = "golang"  
version = "1.20"
```

```
[envs.prod]  
replicas = 70
```

# Что под капотом?



app.toml



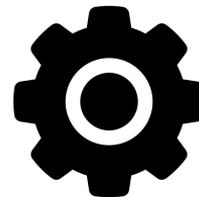
CI



app.toml



YAML



manifest generator



# Этап 3: app.toml

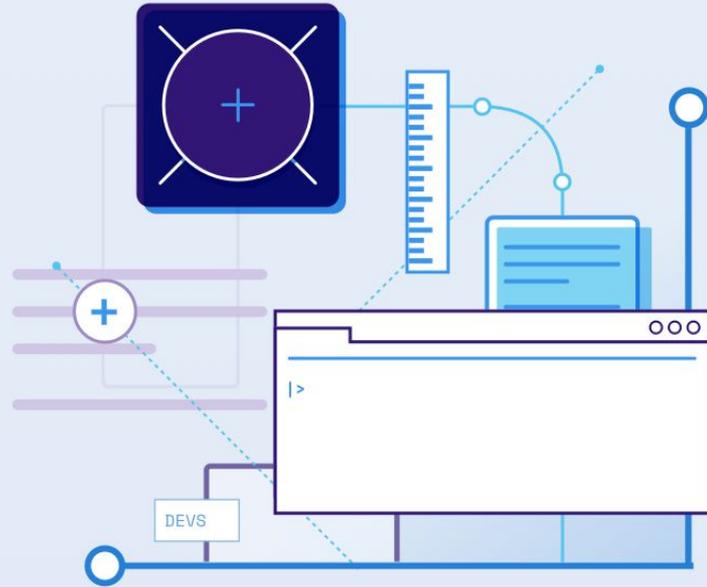
1. Порог входа минимальный
2. Инциденты, связанные с конфигурированием, исчезли как класс
3. Есть возможность в единой точке менять логику формирования манифестов

## Минусы:

1. Новый критичный компонент manifest generator
2. Сложность тестирования генератора

# Что есть готового в той же парадигме?

Open  
Application  
Model



# ОАМ – спецификация

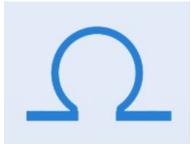
1. Формирует терминологию Application уровня, не k8s сущностей
2. Покрывает цикл разработки от конфигурирования на стыке с платформой до выкатки в production со всеми необходимыми параметрами и особенностями.
3. Не предполагает какой-то конкретной реализации, можно использовать свои стандартные инфраструктурные инструменты / компоненты.

# Kubevela

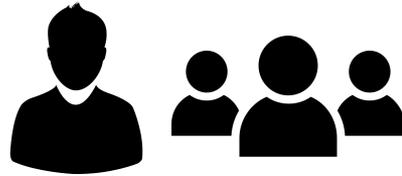


Реализация OAM

# Kubevela

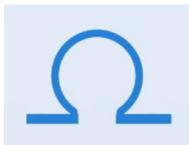


Реализация OAM

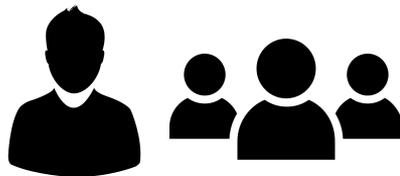


Platform и end  
users

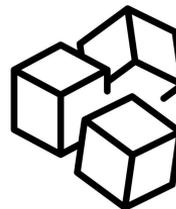
# Kubevela



Реализация OAM

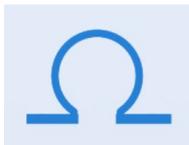


Platform и end  
users

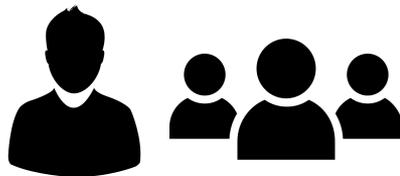


Плагинизируемость  
и гибкость

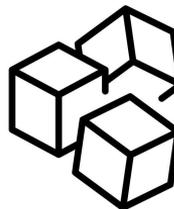
# Kubevela



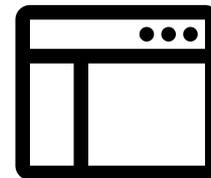
Реализация OAM



Platform и end  
users



Плагинизируемость  
и гибкость



Dashboard / CLI и  
визуализация

# Ключевые особенности

1. Подключение необходимых инструментов, расширяемость
2. Гибкость в настройке для platform team
3. Возможность ограничений для end users
4. Поддержка любых workflow с multicloud/multiclustor/multienv средами

# Application концепт

```
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: <name>
spec:
```

```
  components:
    - name: <component name>
      type: <component type>
      properties:
        <parameter values>
      traits:
        - type: <trait type>
          properties:
            <traits parameter values>
    - name: <component name>
      type: <component type>
      properties:
        <parameter values>
```

```
  policies:
    - name: <policy name>
      type: <policy type>
      properties:
        <policy parameter values>
  workflow:
    - name: <step name>
      type: <step type>
      properties:
        <step parameter values>
```

components

Описывает ключевые составные части приложения: API, workers, cron, database и подобные

# Application концепт

```
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: <name>
spec:
```

```
  components:
    - name: <component name>
      type: <component type>
      properties:
        <parameter values>
        traits:
          - type: <trait type>
            properties:
              <traits parameter values>
    - name: <component name>
      type: <component type>
      properties:
        <parameter values>
```

```
  policies:
    - name: <policy name>
      type: <policy type>
      properties:
        <policy parameter values>
  workflow:
    - name: <step name>
      type: <step type>
      properties:
        <step parameter values>
```

components

Описывает ключевые составные части приложения: API, workers, cron, database и подобные

traits

Описывает настройки управления сущностью: количество реплик, стратегия выкатки, внешний discovery, ingress

# Application концепт

```
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: <name>
spec:
  components:
    - name: <component name>
      type: <component type>
      properties:
        <parameter values>
      traits:
        - type: <trait type>
          properties:
            <traits parameter values>
    - name: <component name>
      type: <component type>
      properties:
        <parameter values>
  policies:
    - name: <policy name>
      type: <policy type>
      properties:
        <policy parameter values>
  workflow:
    - name: <step name>
      type: <step type>
      properties:
        <step parameter values>
```



policies

Те же traits, только  
глобальные.  
Топология, в какие  
кластера катиться, security  
политики и тд

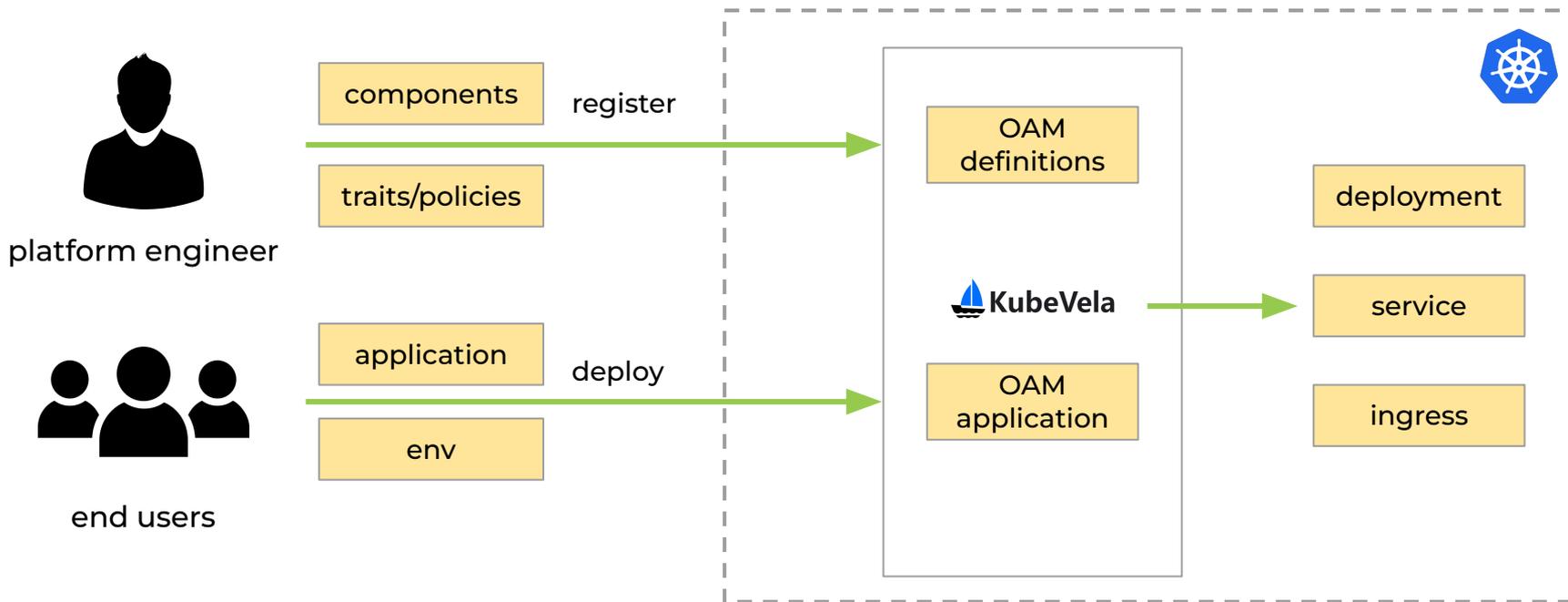
# Application концепт

```
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: <name>
spec:
  components:
  - name: <component name>
    type: <component type>
    properties:
      <parameter values>
    traits:
    - type: <trait type>
      properties:
        <traits parameter values>
  - name: <component name>
    type: <component type>
    properties:
      <parameter values>
  policies:
  - name: <policy name>
    type: <policy type>
    properties:
      <policy parameter values>
  workflow:
  - name: <step name>
    type: <step type>
    properties:
      <step parameter values>
```

workflow

Описывает набор шагов  
для деплоя или любого  
другого процесса вокруг  
Application

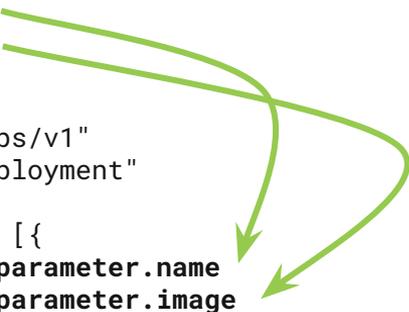
# Концепция: разделение зон ответственности



# А что под капотом?

# Definitions

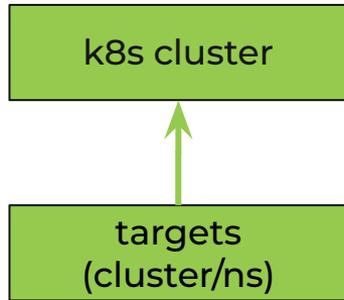
```
webserver: {  
  type: "component"  
  attributes: {}  
}  
  
template: {  
  parameter: {  
    name: string  
    image: string  
  }  
  output: {  
    apiVersion: "apps/v1"  
    kind: "Deployment"  
    spec: {  
      containers: [{  
        name: parameter.name  
        image: parameter.image  
      }]  
    }  
  }  
}
```



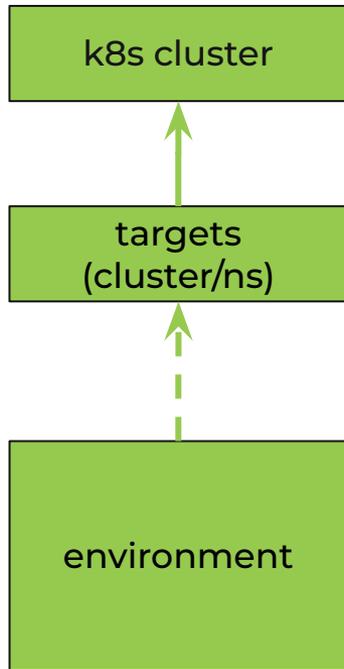
# Application

```
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: first-vela-app
spec:
  components:
    - name: express-server
      type: webserver
      properties:
        image: oamdev/hello-world
  traits:
    - type: scaler
      properties:
        replicas: 1
```

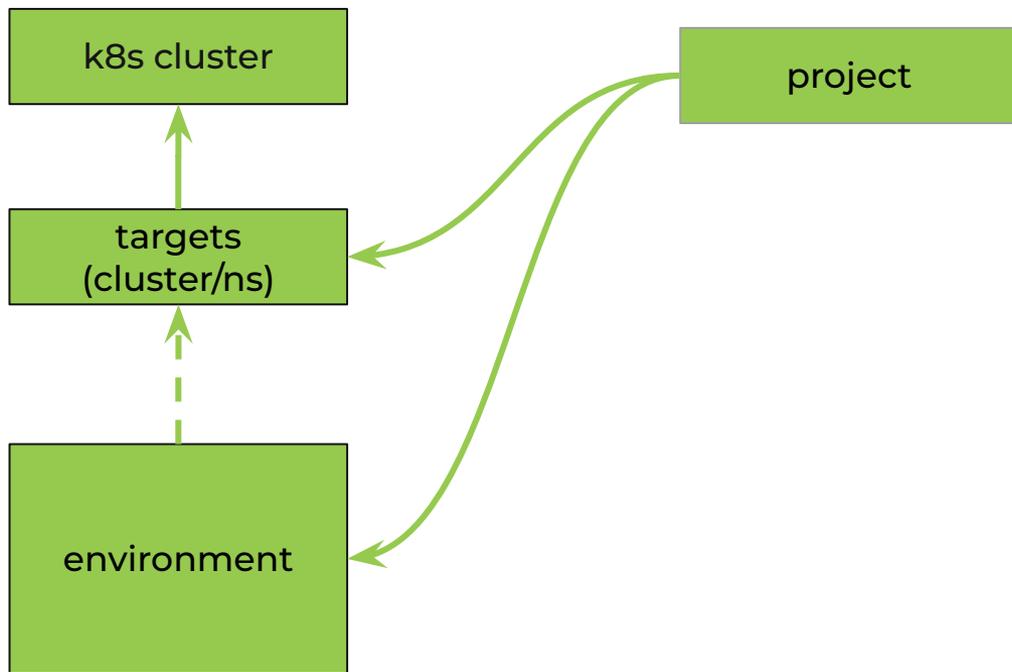
# Суццности kubevela



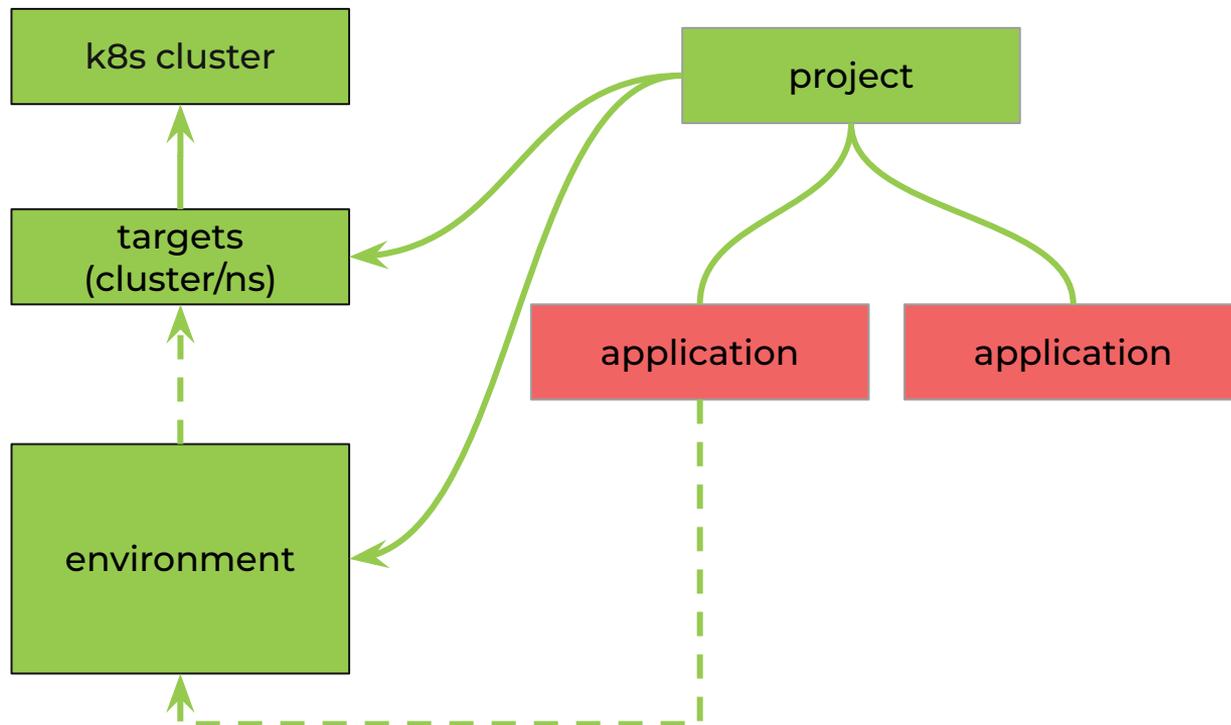
# Сущности kubernetes



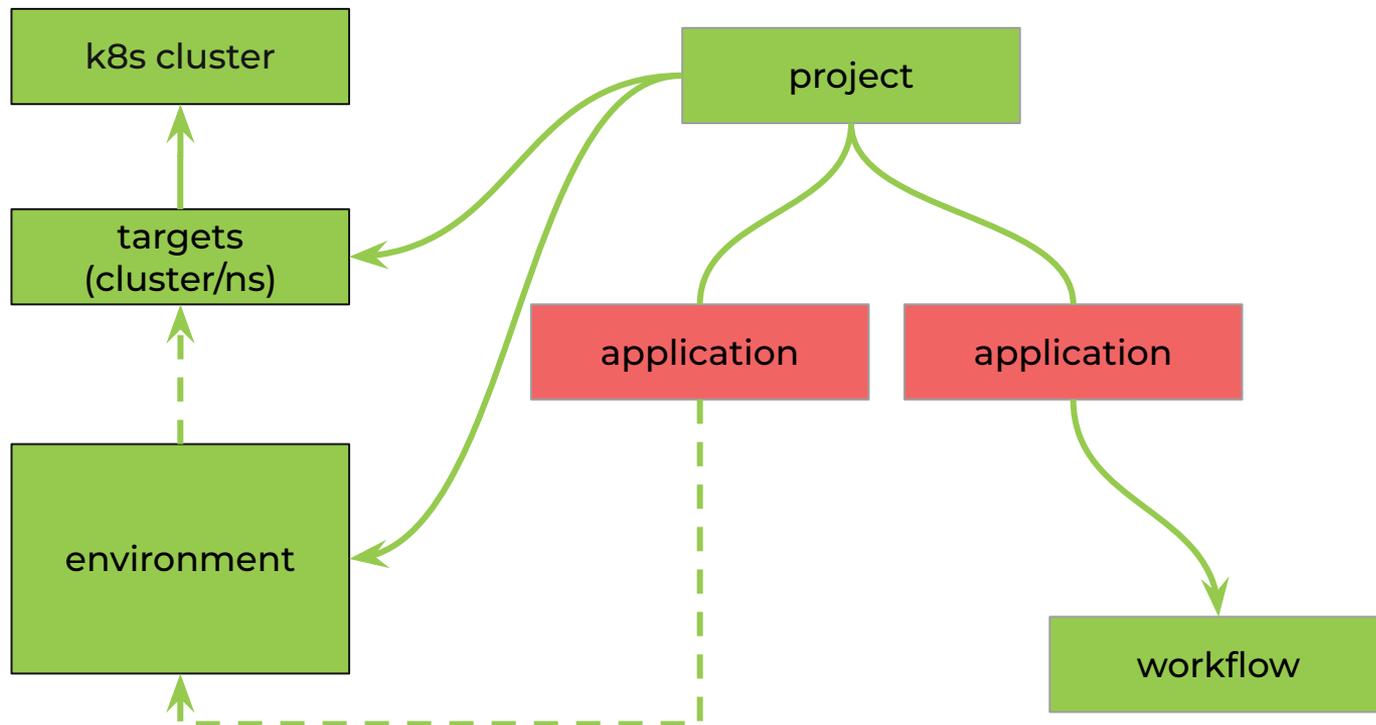
# Сущности kubernetes



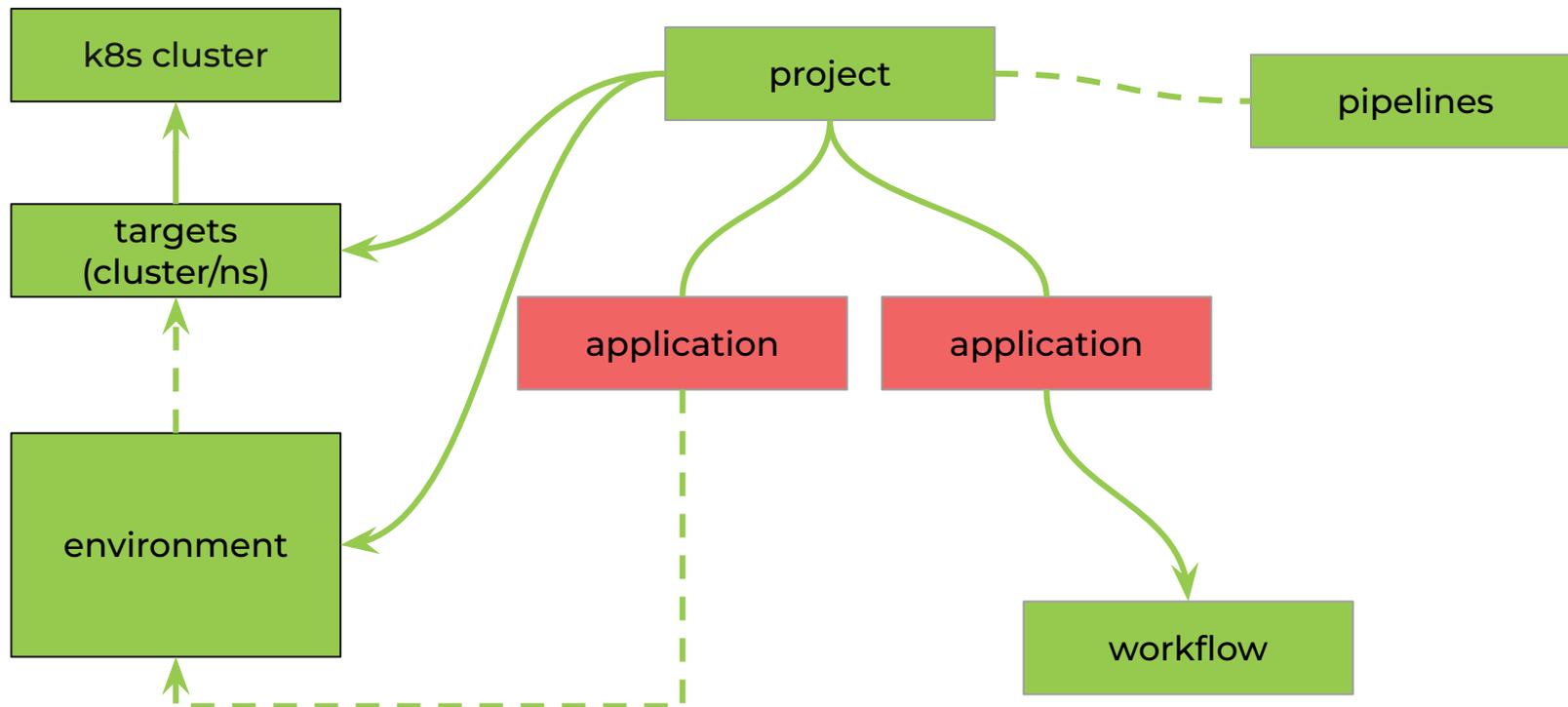
# Сущности kubernetes



# Суццности kubernetes



# Суццности kubernetes



# Система addons

```
|— resources/  
|   |— xxx.cue  
|   |— xxx.yaml  
|— definitions/  
|— schemas/  
|— config-templates/  
|   |— xxx.cue  
|— views/  
|   |— xxx.cue  
|— README.md  
|— metadata.yaml  
|— parameter.cue  
|— template.yaml(or template.cue)
```

# Старт с kubevela

```
$ vela install
```

```
... 
```

```
$ vela addon enable velaux
```

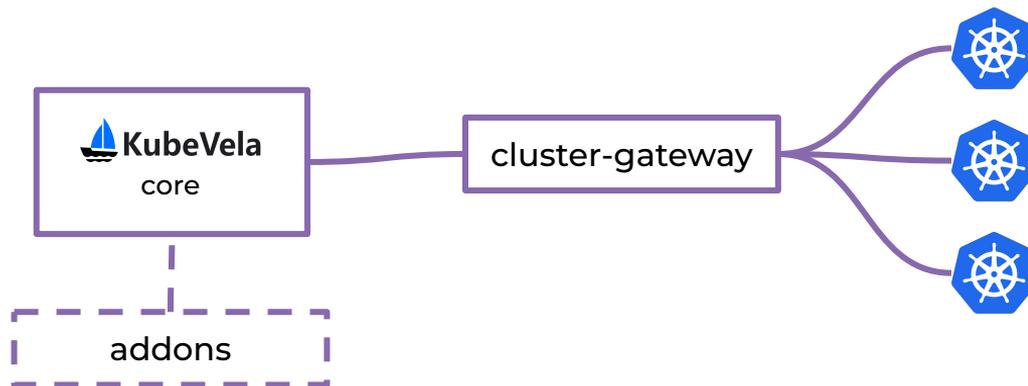
```
Addon velaux enabled successfully.
```

```
Please access addon-velaux from the following endpoints:
```

CLUSTER	COMPONENT	REF(KIND/NAMESPACE/NAME)	ENDPOINT	INNER
local	velaux-server	Service/vela-system/velaux-server	velaux-server.vela-system:8000	true

# Kubevela control plane

vela-system	kubevela-cluster-gateway-98787b79b-wzp4t	1/1	Running	8d
vela-system	kubevela-vela-core-59c769fdcd-hrpmb	1/1	Running	8d
vela-system	velaux-server-869dbbbf7c-slmsl	1/1	Running	8d



# Kubevela со стороны **platform engineer**

KubeVela Continuous Delivery Extension Admin Dashboard Administrator

Clusters Clusters Setup Kubernetes clusters by adding an existing one or creating a new one via cloud provider Connect From Cloud Connect Existing Cluster

Search by Name and Description etc



**local** Local

The hub manage cluster where KubeVela runs on.

• Healthy

## Kubevela UX

# Инфраструктурный СТЫК



# Подготовка к выдаче инструмента разработчикам



# Kubevela со стороны конечного пользователя

# Создание Application и релизный цикл

- Applications
- Pipelines
- Environments
- Targets
- Projects

### Applications Deploy and manage all your applications

New Application

Search by Project Search by Environment Search by Name and Description Search by Label Selector

Name(Alias)	Project	Description	Labels	Actions
addon-velaux(addon-velaux)	system	Automatically converted from KubeVela Application in Kubernetes.		<a href="#">Edit</a> <a href="#">Remove</a>

# Релизный цикл

KubeVela Workflow

# Итоги

## Преимущества:

- Предоставление понятного и удобного интерфейса пользователям
- Гибкость и расширяемость платформы заложена архитектурно
- TTM ускоряется, количество инцидентов падает

## Недостатки:

- Сложность в поиске баланса между простотой и гибкостью
- Тестирование такого подхода – непростая задача
- Реализация kubevela пока что сырая и требует доработки

# Вопросы

# avito.tech

Москва — 2023

# Александр Лукьянченко

Руководитель  
разработки PaaS

 [@lookyan](https://t.me/lookyan)