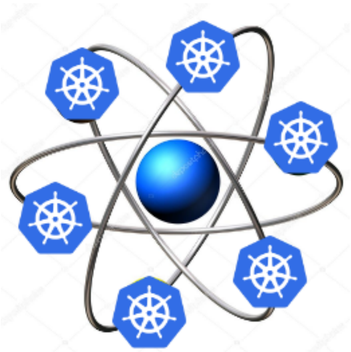
Как мы строили систему управления Kubernetes кластерами



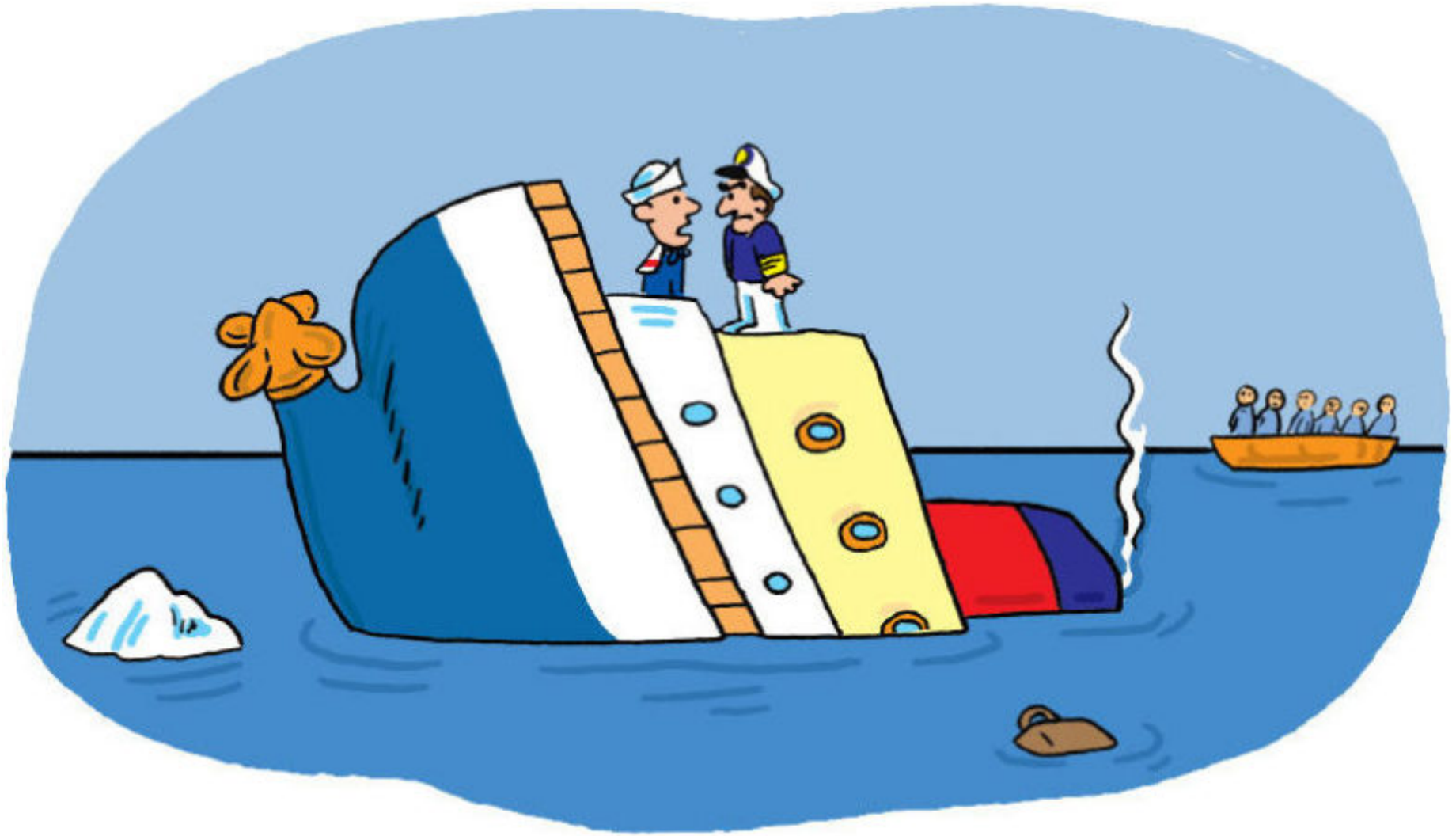
Alena Prokharchyk, Principal Software Engineer

@RancherLabs


ВОТ ТАК, С ПОМОЩЬЮ НЕХИТРЫХ ПРИСПОСОБЛЕНИЙ
БУХАНКУ БЕЛОГО (ИЛИ ЧЕРНОГО) ХЛЕБА
МОЖНО ПРЕВРАТИТЬ В ТРОЛЛЕЙБУС...
НО ЗАЧЕМ?!
УМЕЛЫЕ РУКИ.

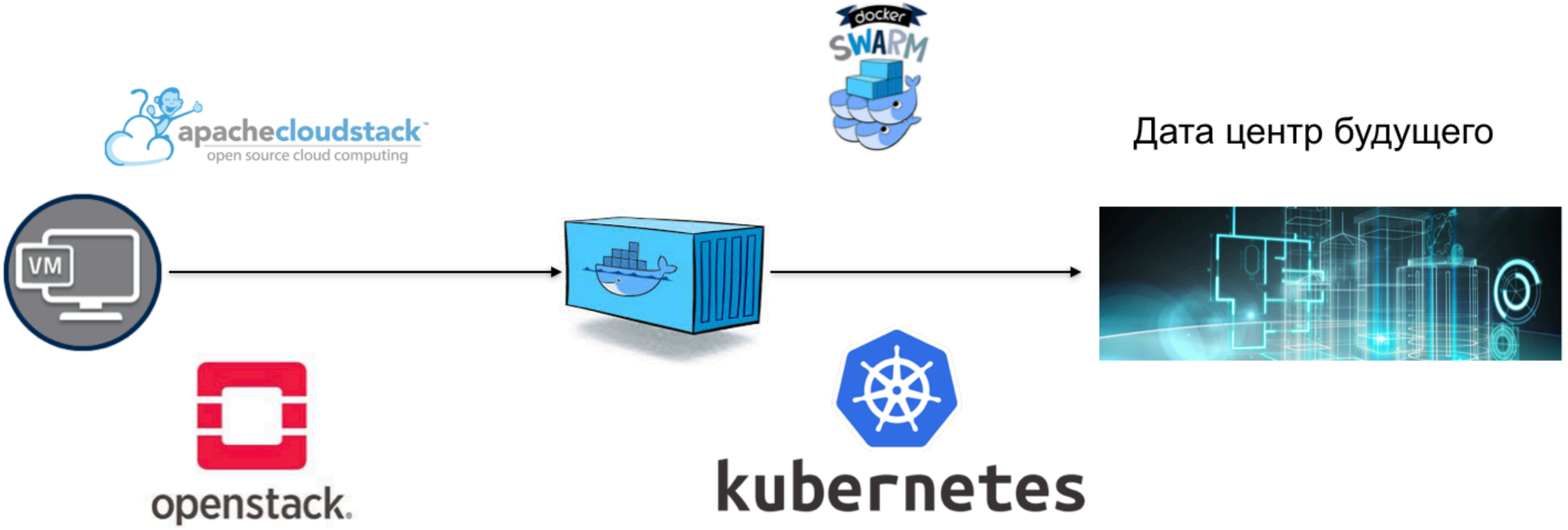
КАК

Учимся на ошибках



OK ... so "full speed astern" means "stop." Got it.

Немного истории



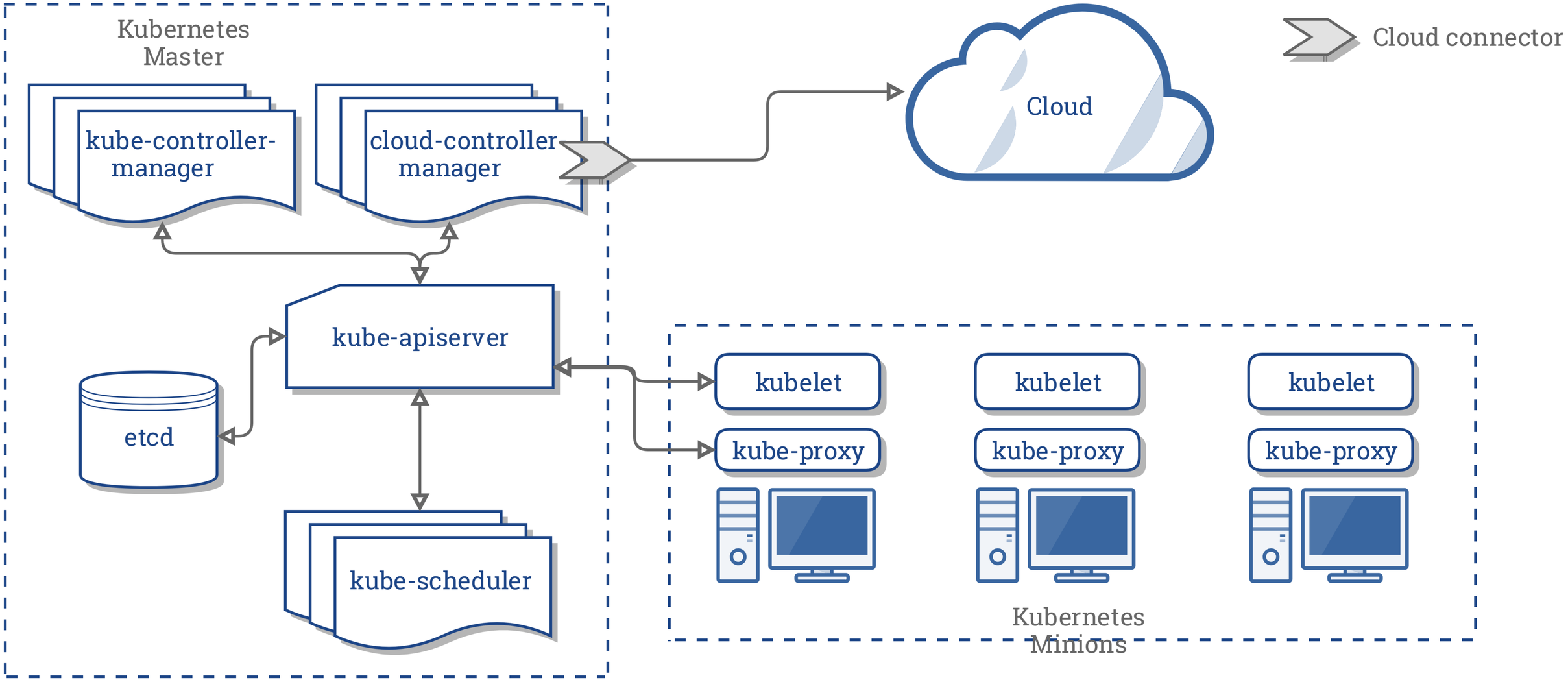
Для чего нужен оркестратор

- Масштабировать и запускать контейнеры на (n) хостах
- Балансировать нагрузку
- Помогать с live апгрейдами приложения

kubernetes

Самый популярный оркестратор для контейнеров

Никто не сказал, что будет легко…



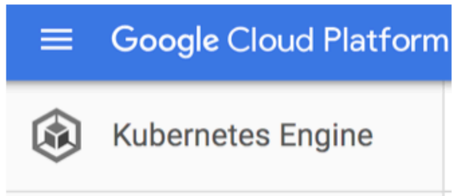
Запустить оркестратор - работа еще та

- Инсталляция была сложная
- Апгрейд был еще сложнее
- 100 инструкций для разных типов установок

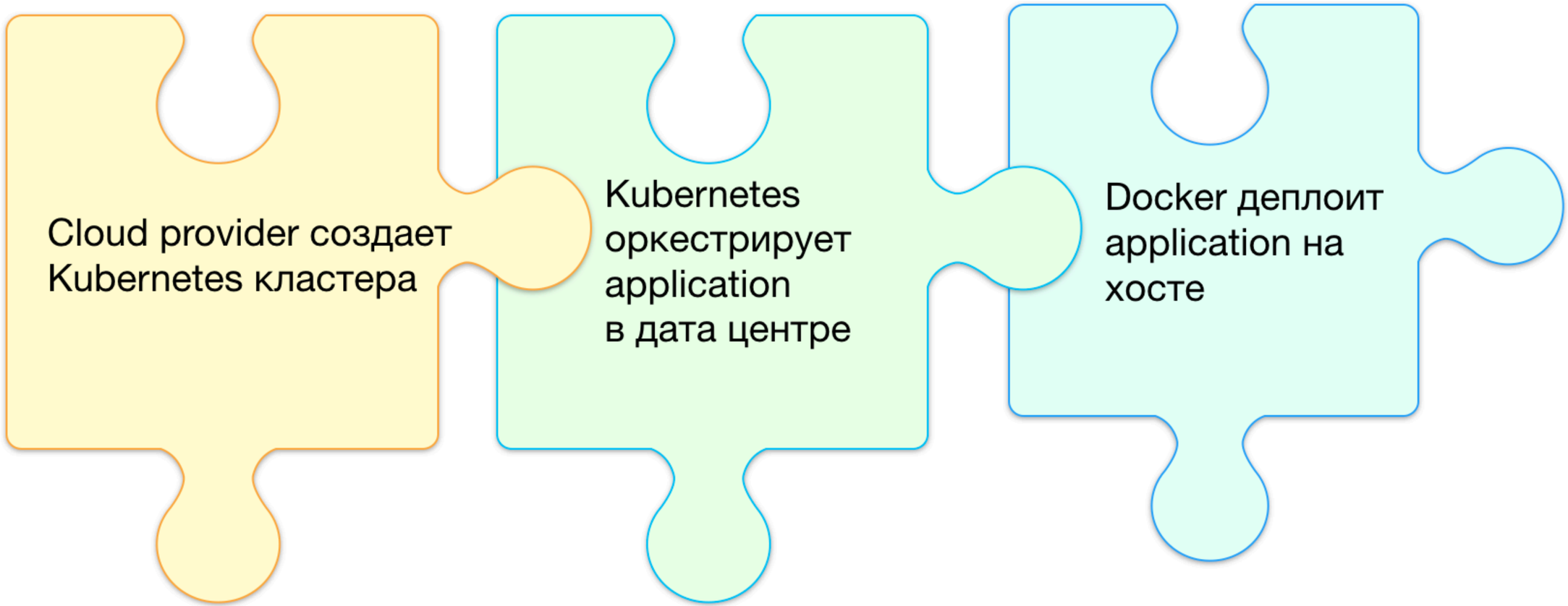
Со временем стало полегче

Появилось куча автоматизированных инсталляторов

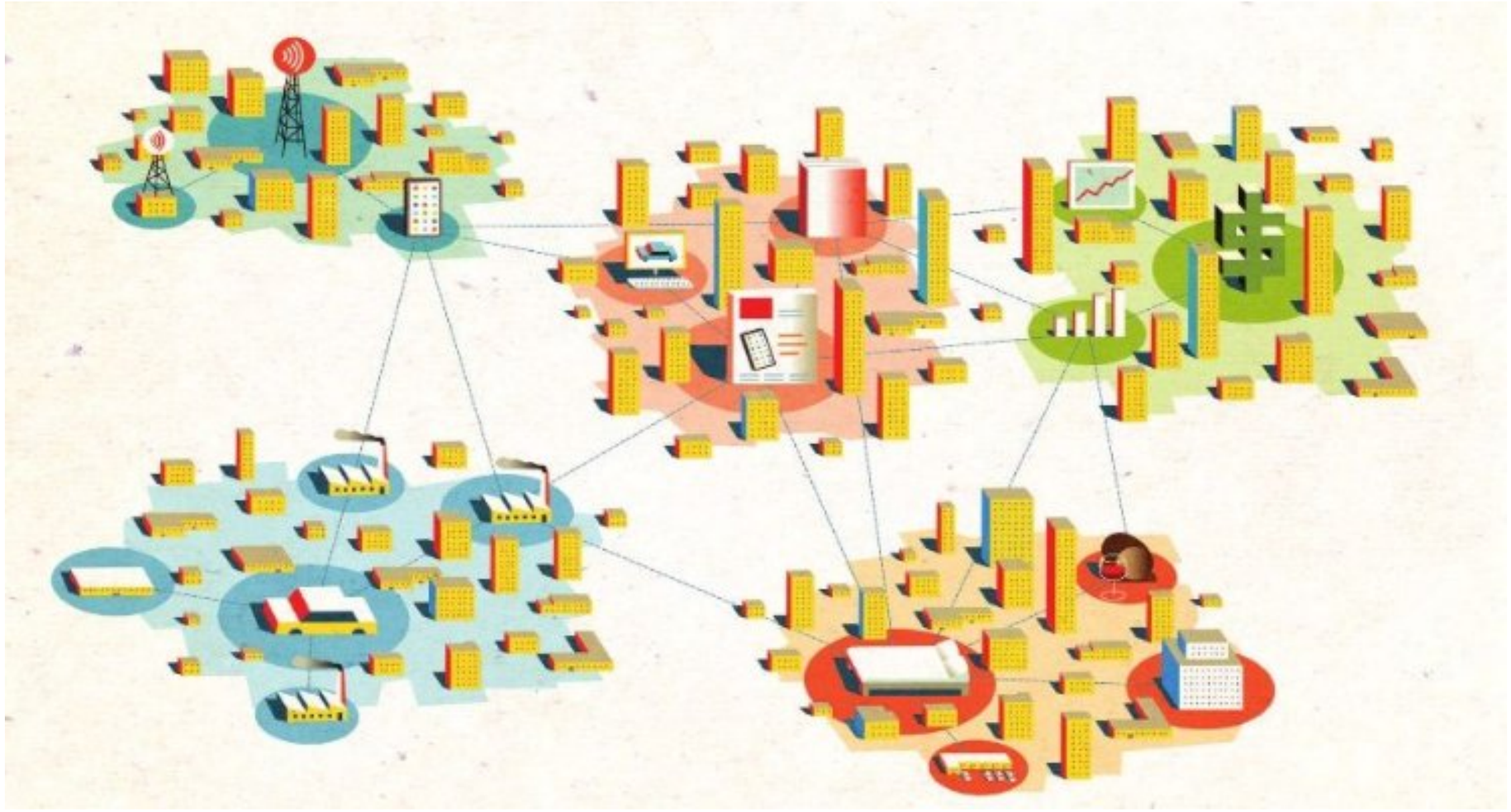
Плюс “Kubernetes как сервис” в облаках



Пазл сложился, работы больше нет?

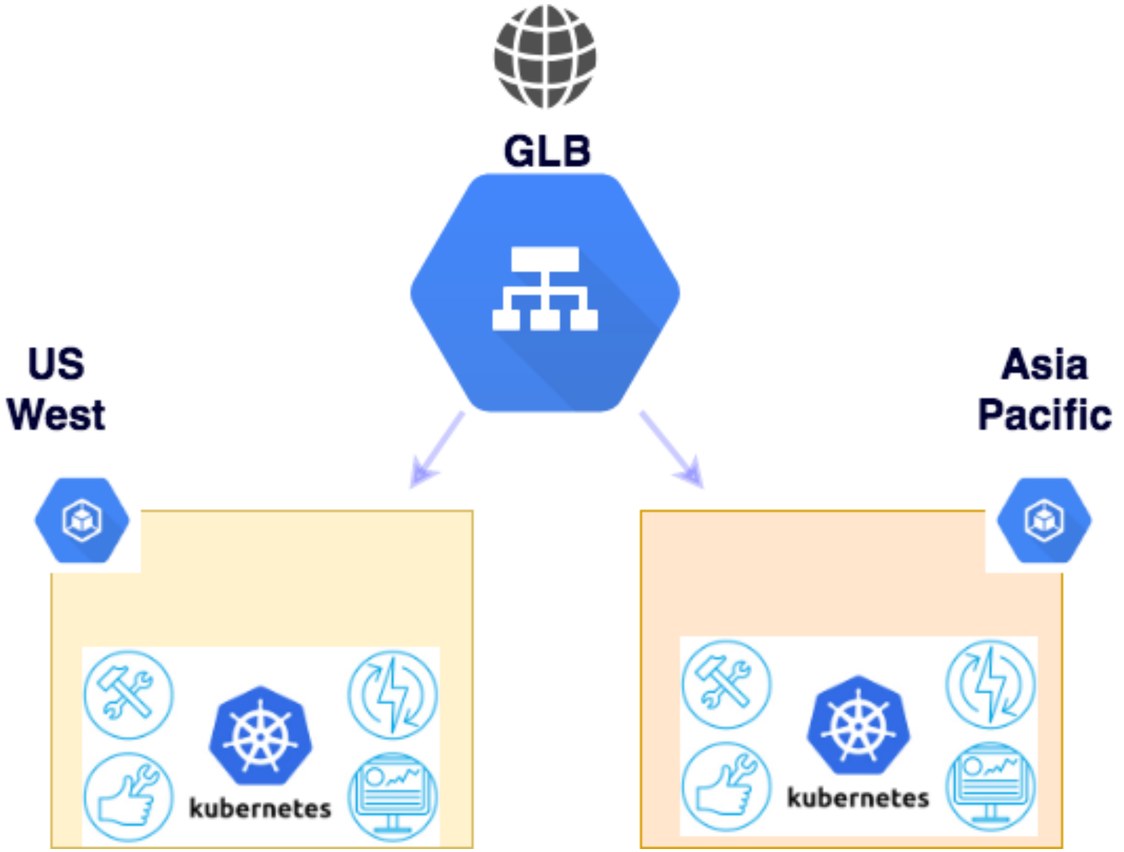


Зачастую вам нужно несколько кластеров



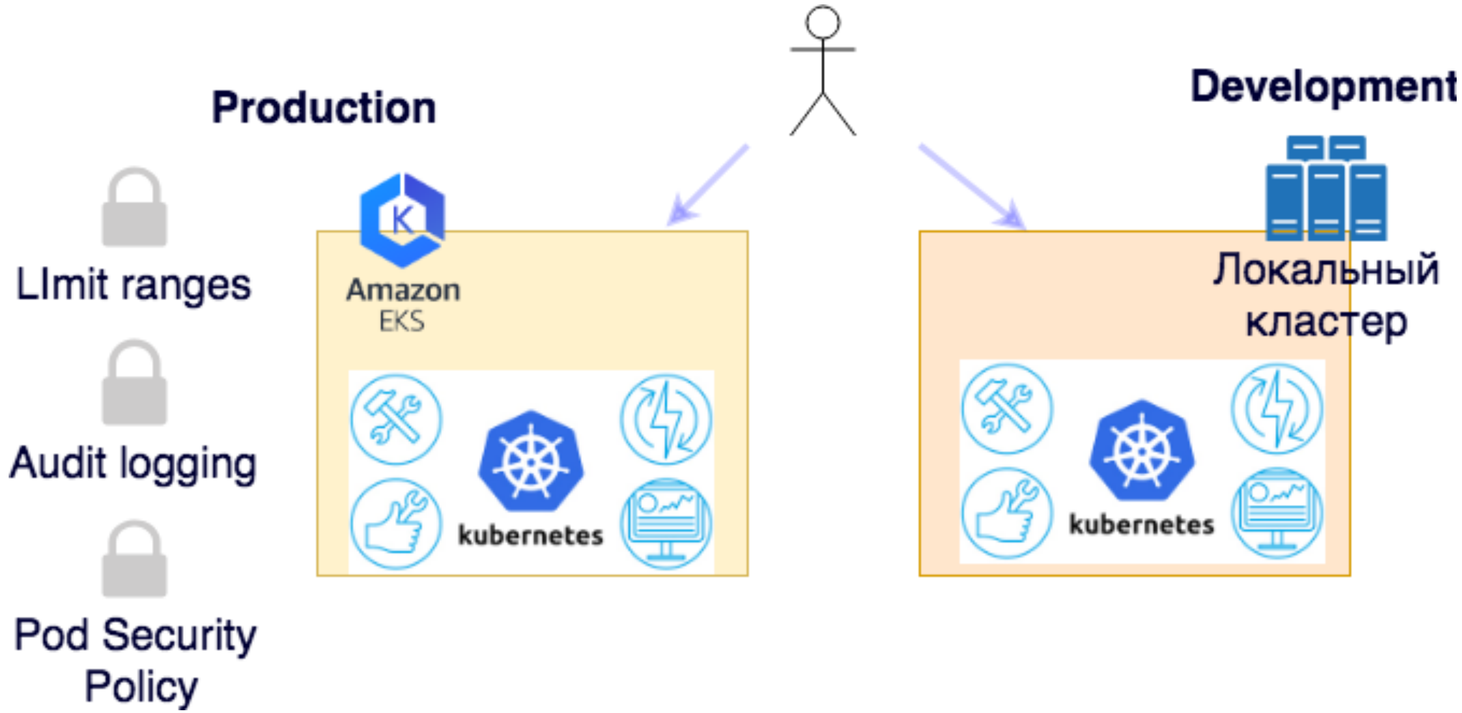
Пример #1 - Географическое разделение

- Кластер на регион
- Глобальная балансировка нагрузки
- Компоненты Kubernetes близко друг к другу



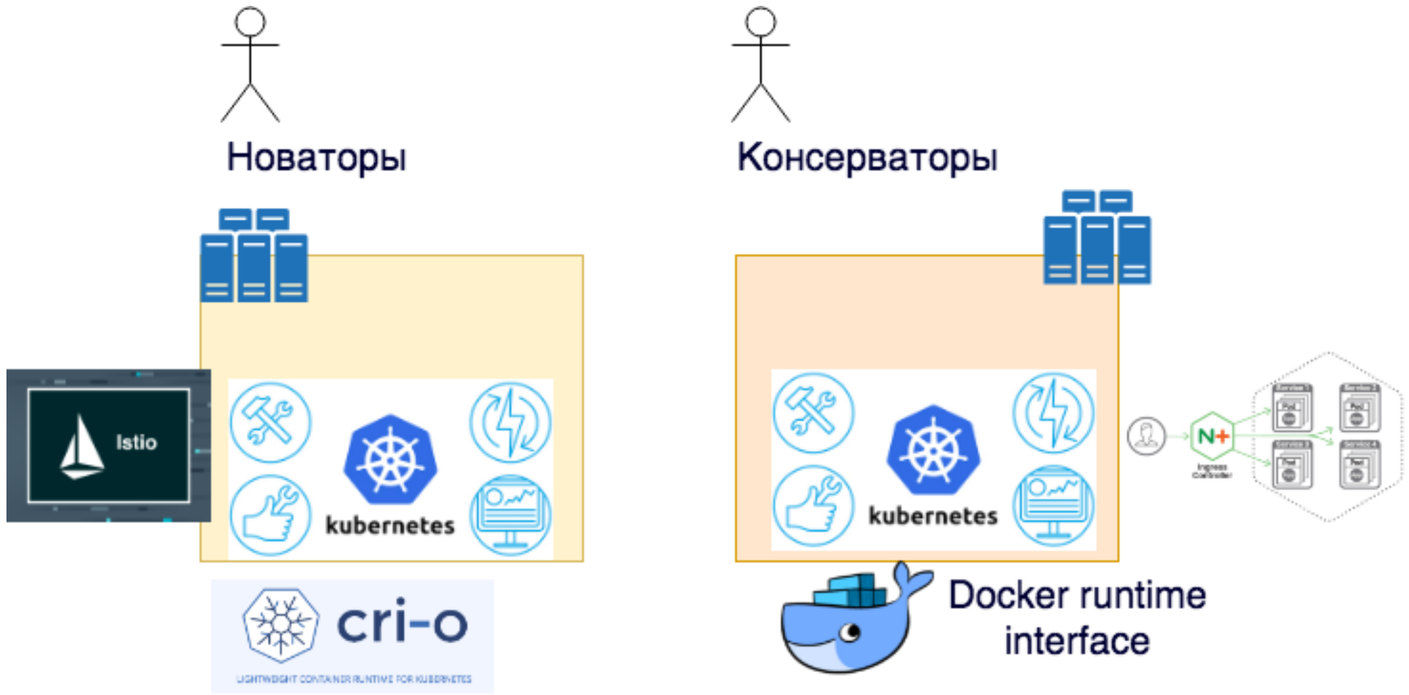
Пример #2 - Логическое разделение по принципу безопасности

- Кластер на проект
- Разная степень защиты



Пример #3 - Логическое разделение по функциональному принципу

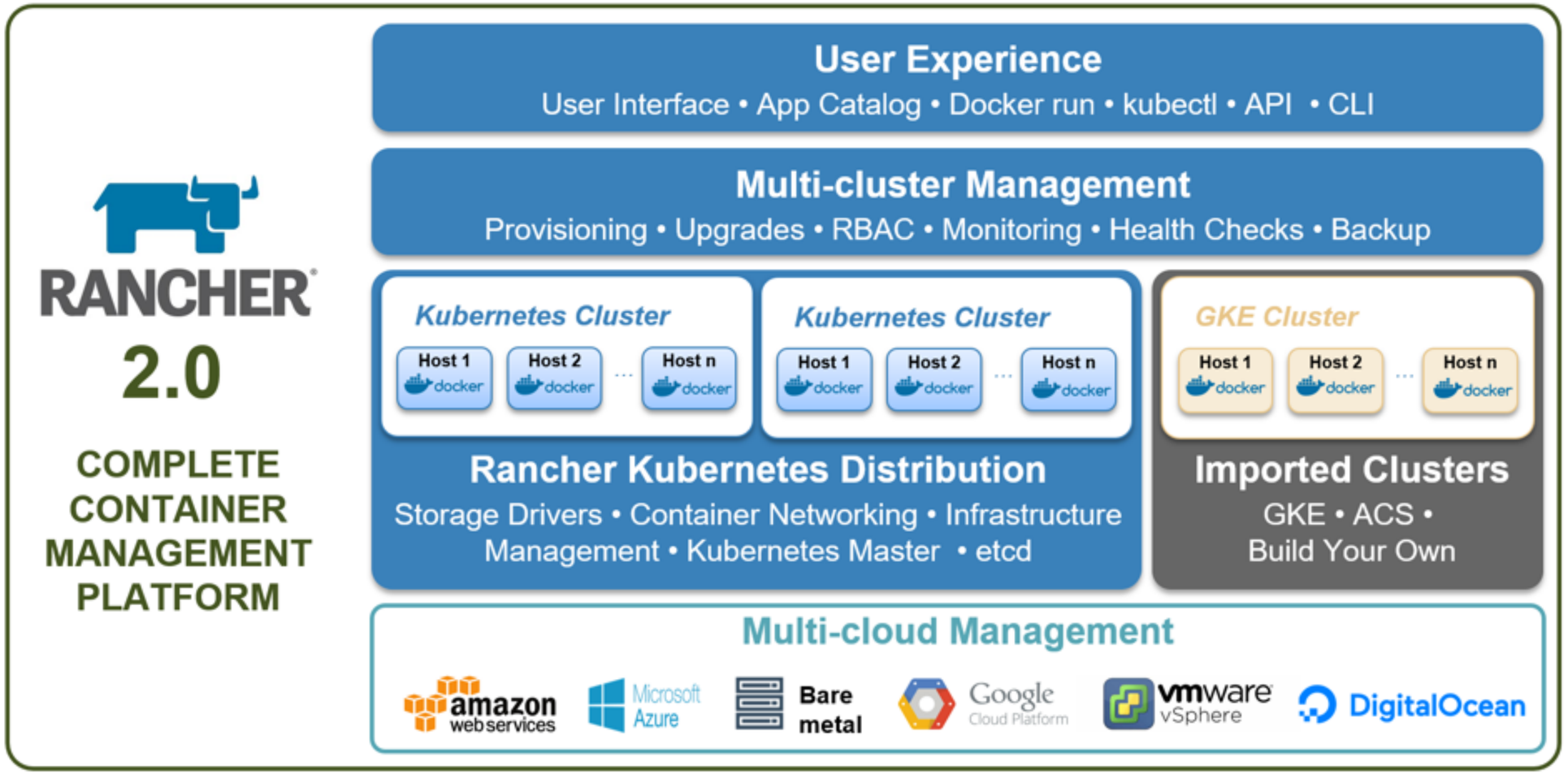
- Кластер на команду
- Разные команды = разные лучшие практики



Что надо было сделать

- **Централизованный** Kubernetes инсталлятор
- **Централизованный** доступ к кластерам
- **Централизованный** администратор аддонов

…мы строили, строили, и наконец построили Open source систему управления Kubernetes кластерами



Проблема #1 - Инсталляция

Требования к инсталляции

- Создавать кластер как сервис в облаке
- Создавать виртуалки в облаке, и разворачивать там кластер
- Разворачивать кластер на существующих хостах

Муки выбора инсталлятора Kubernetes

- **kops** работает только в одном облаке
- **kubeadm**, **kubespray** были рассмотрены и отметены по ряду проблем

Решение: написать свой инсталлятор, который:

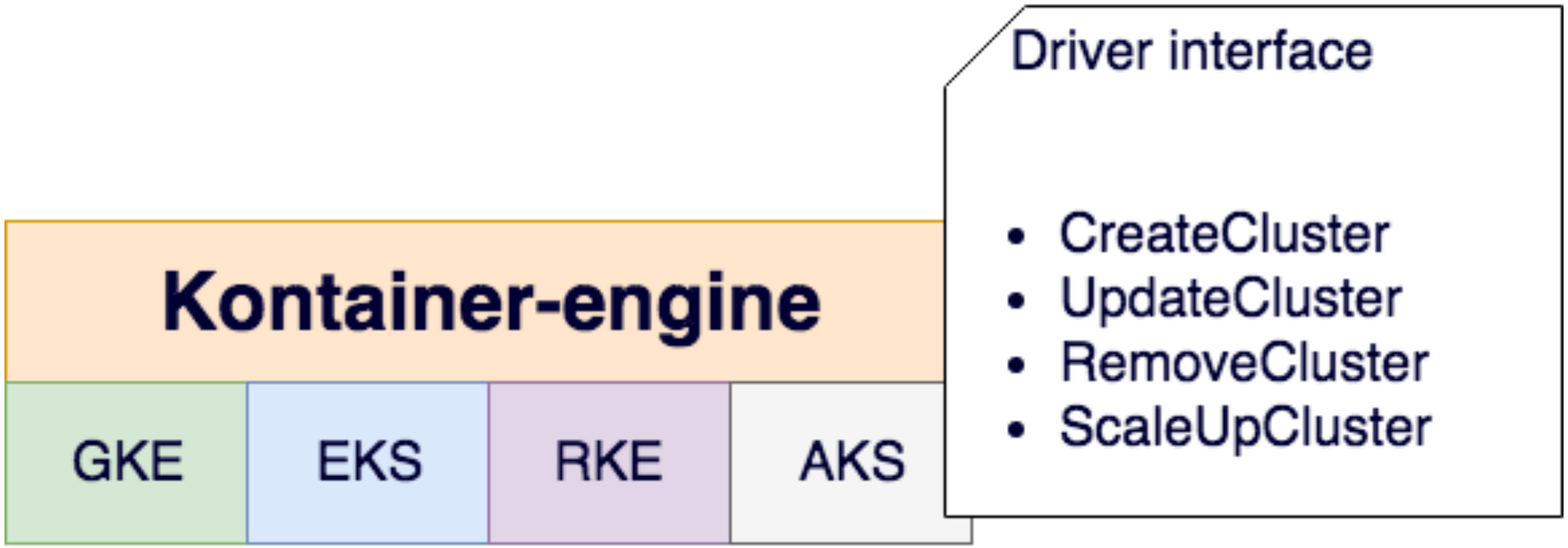
- Разворачивает кластер в облаке, или в локальном дата центре
- Запускается из командной строки, или в интеграции
- Запускает Kubernetes в Docker контейнерах
- С самой простой в мире конфигурацией
- Написан на Go

RKE
как автономный инсталлятор

Rancher
Kubernetes Engine

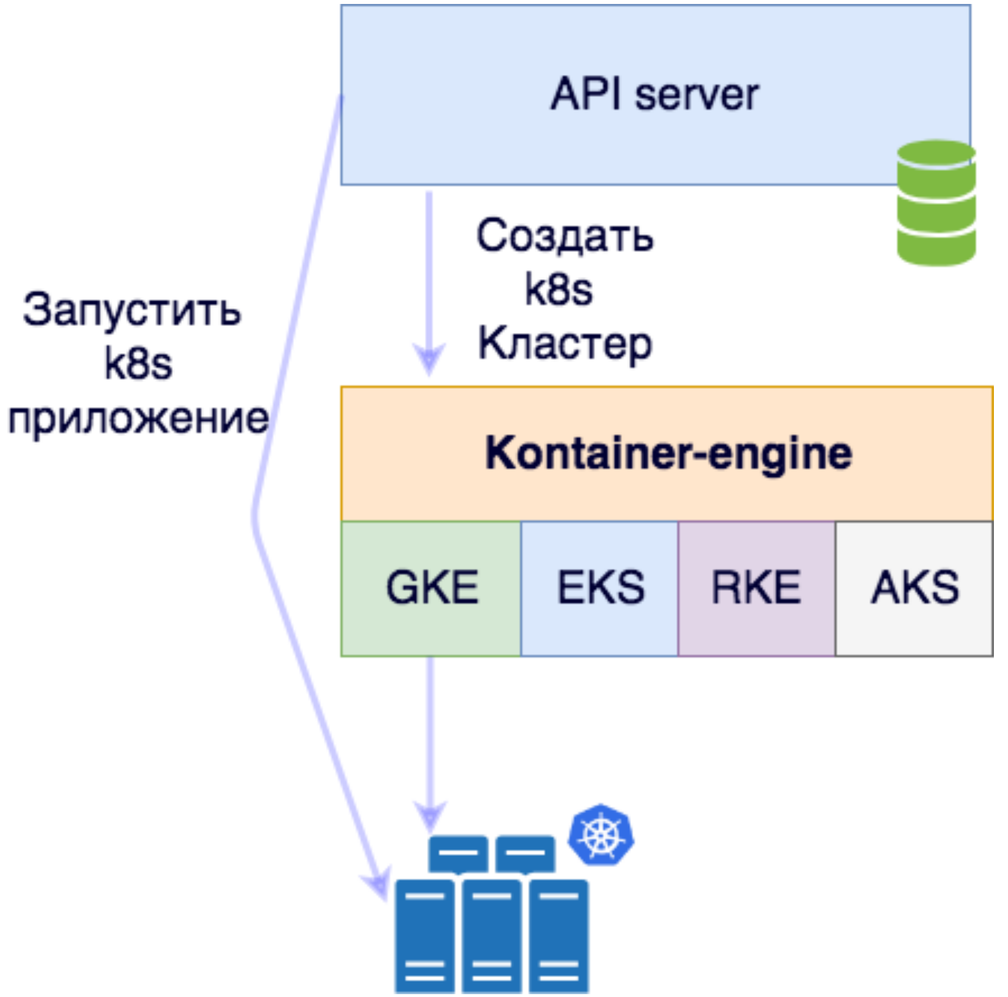
https://github.com/rancher/rke

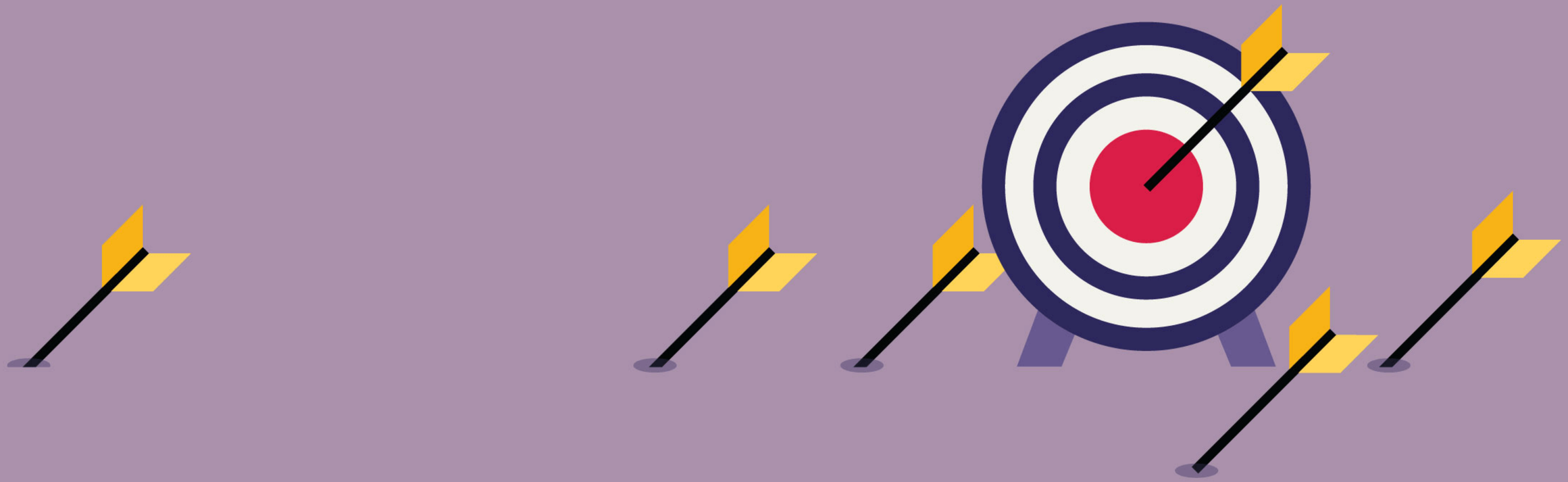
RKE как драйвер



API/Management Framework

- Пользователь создает кластер через API
- Kontainer-engine осуществляет установку, и возвращает ключи доступа
- Сервер подсоединяется к кластерам, используя ключи





Rancher 2.0, версия 1

- Написана на Java, с Mysql DB - по следам Rancher 1.x
- Реализовано представление объектов Kubernetes через Rancher API
- Предпринята попытка поддержать 2 модели создания объектов: через Rancher API и нативно, через kubectl

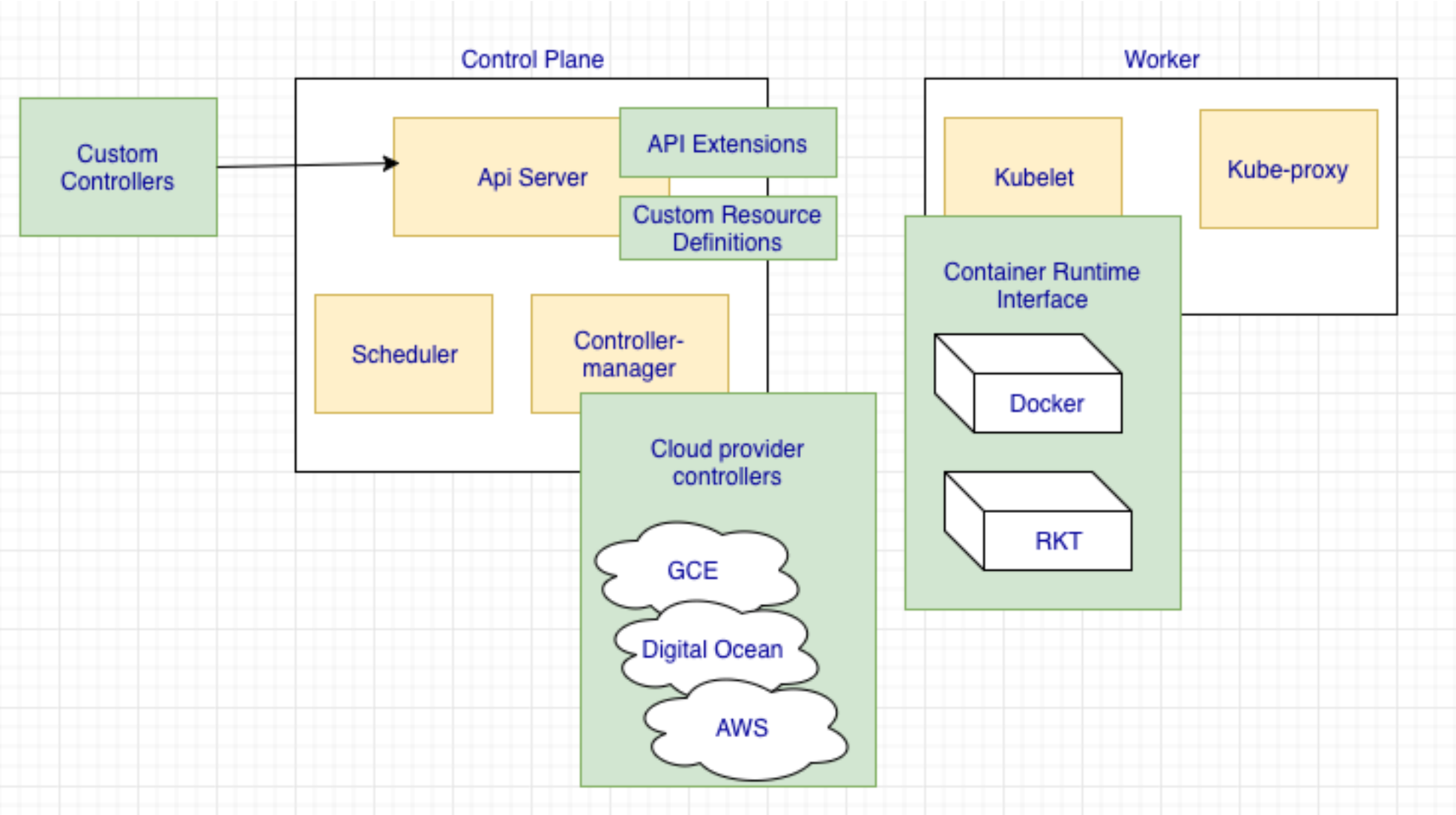
Дизайн был ошибочен

- Много конвертирования из Java в Go для работы с Docker, Kubernetes, RKE, Kontainer-engine
- Большая разница в архитектуре кода для коммуникации с кластерами и слоем менеджмента
- Куча усилий и багов в попытке поддержки 2-х моделей создания объектов

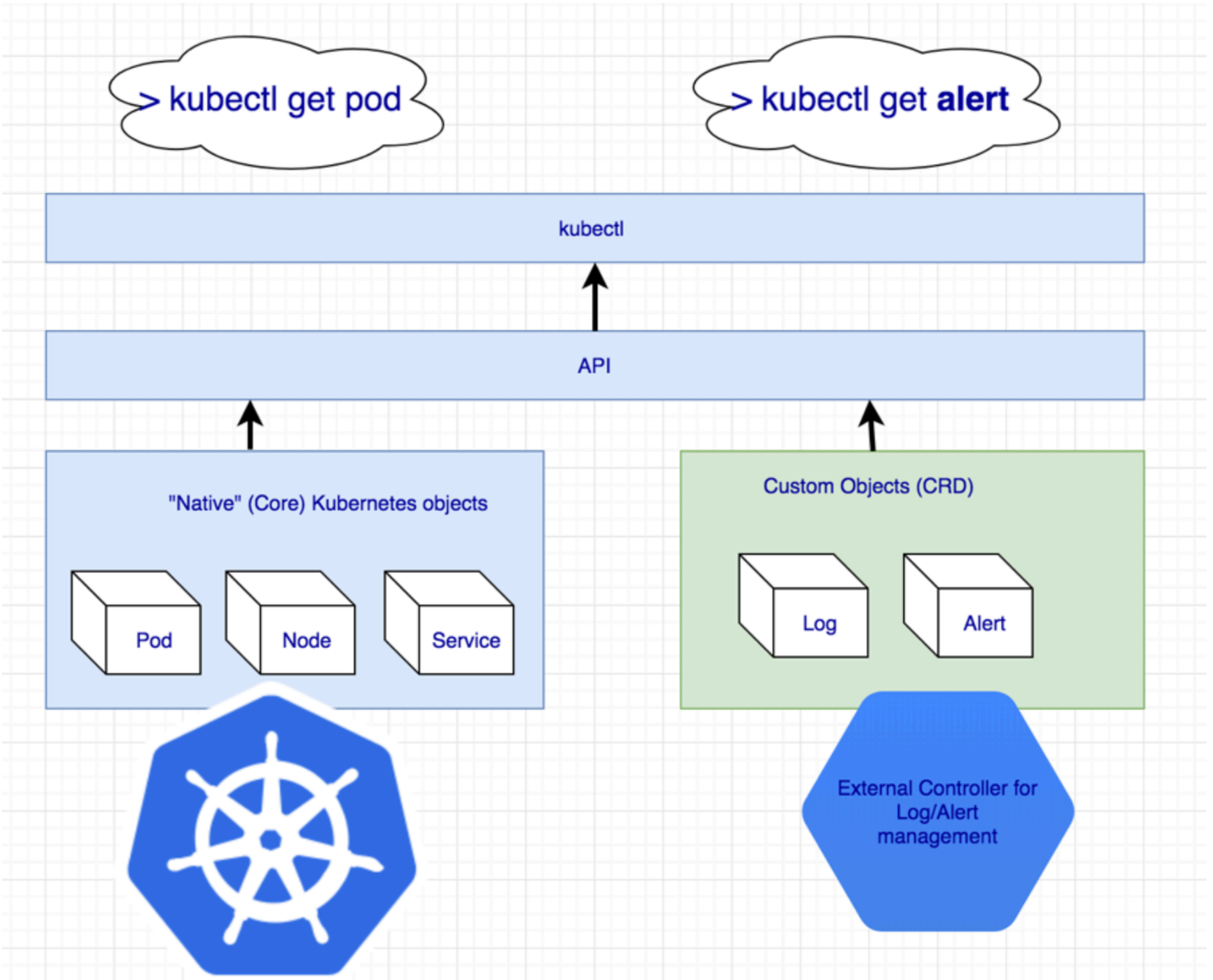
Rancher 2.0, версия 2

- Построен на базе Kubernetes, с базой данных **etcd**
- Написан на **Go**
- Rancher API расширяет Kubernetes API
- Каждый кастомный объект Rancher - это Kubernetes **CRD** (Custom resource definition)
- Все функциональные компоненты написаны как Kubernetes **controllers**

Kubernetes - это крутая девелоперская платформа



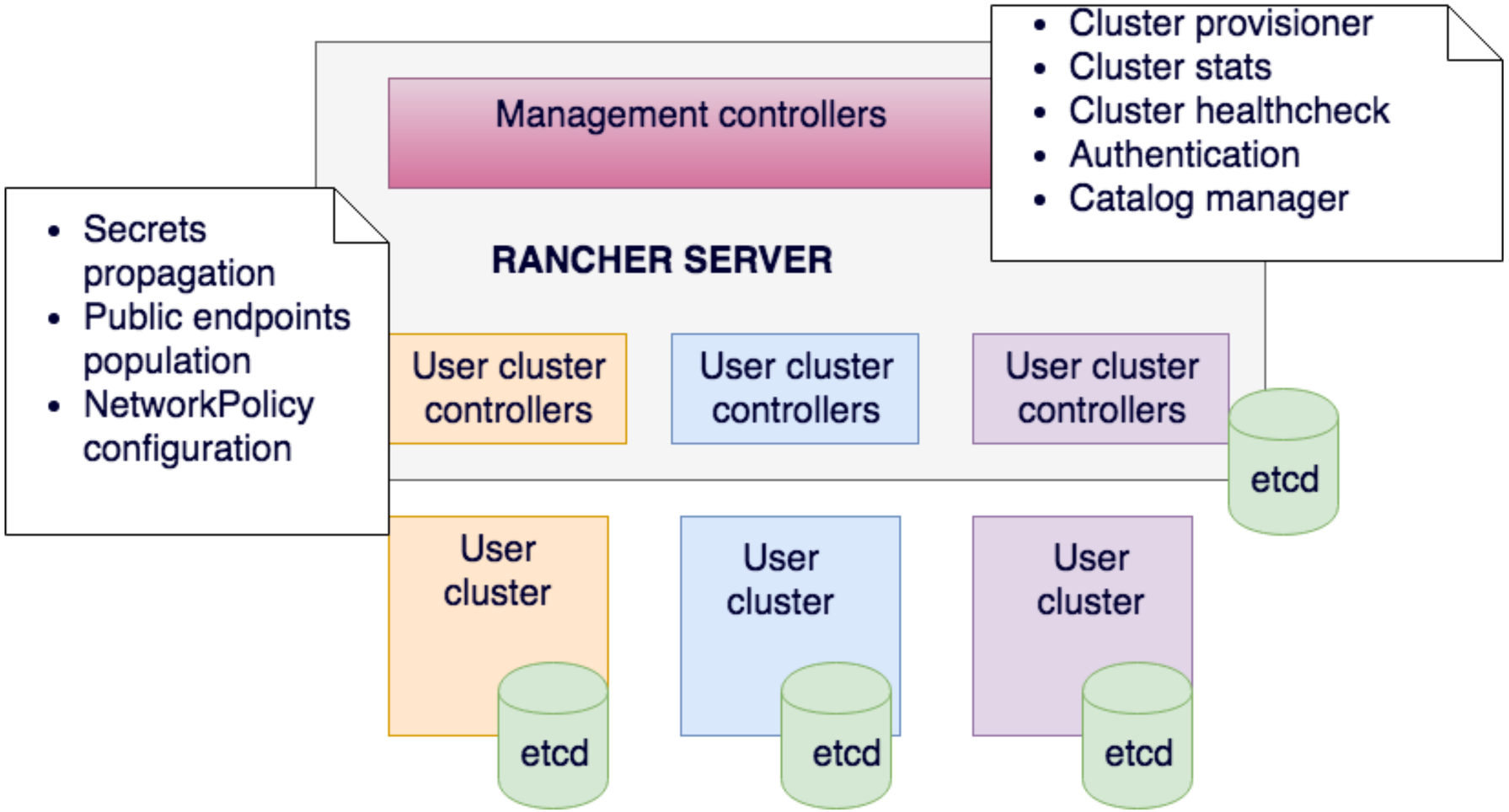
Что такое CRD?



Принципы работы controller

- Controller подписывается в API на события объектов
- Пользователь создает объект через API, и controller получает уведомление.
- Controller выполняет логику и обновляет объект
- Над одним объектом зачастую работают несколько контроллеров.
- Самая популярная custom controller SDK - **client-go**.

Архитектура Rancher с высоты птичьего полета



Работа над ошибками

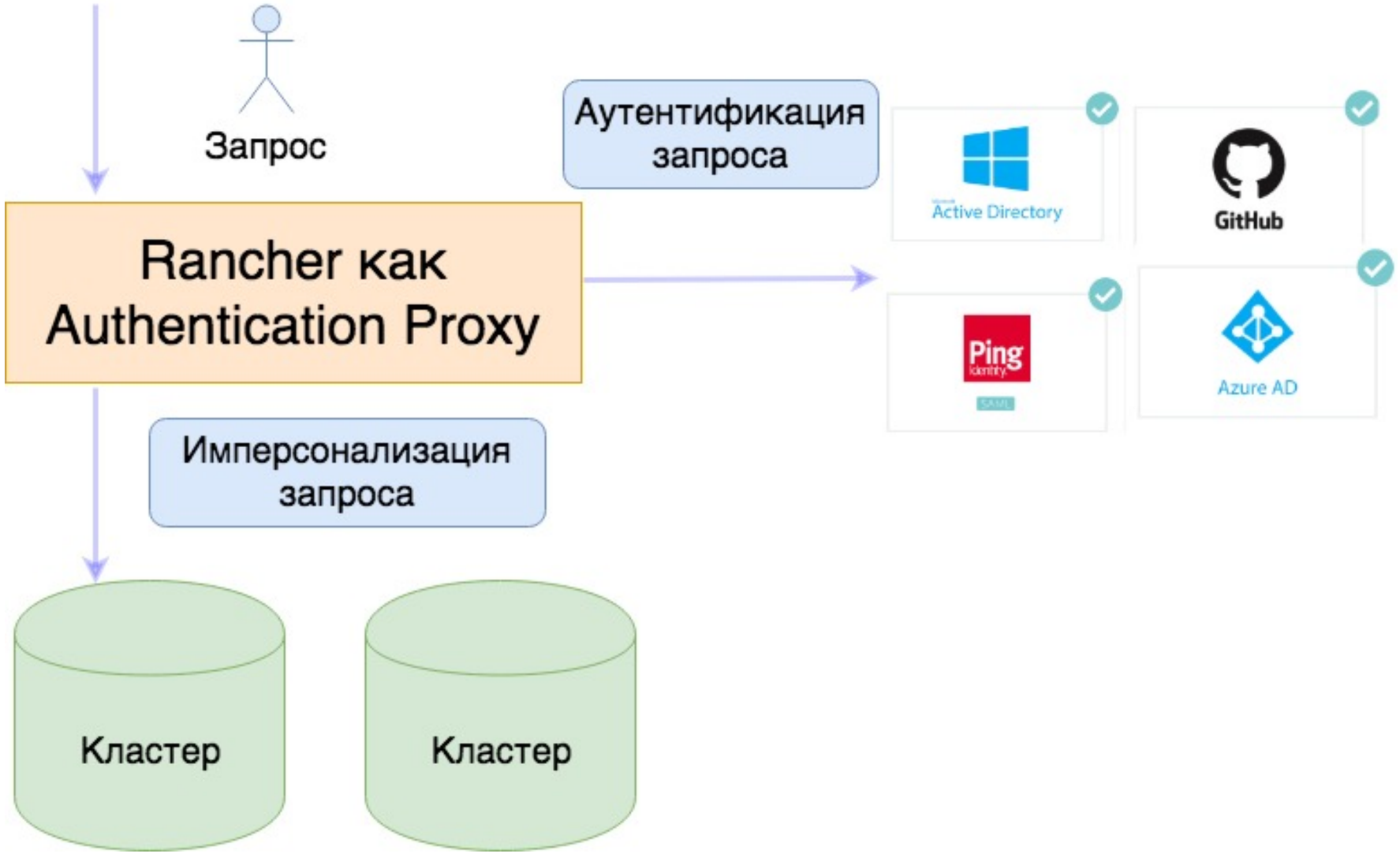
- Быть осторожным, выбирая third party компоненты для ключевой части продукта
- Если возможно, выпускать часть проекта автономно
- Рассмотривать возможность использования языка и технологий системы, с которой происходит интеграция

Проблема #2 - Аутентификация и авторизация

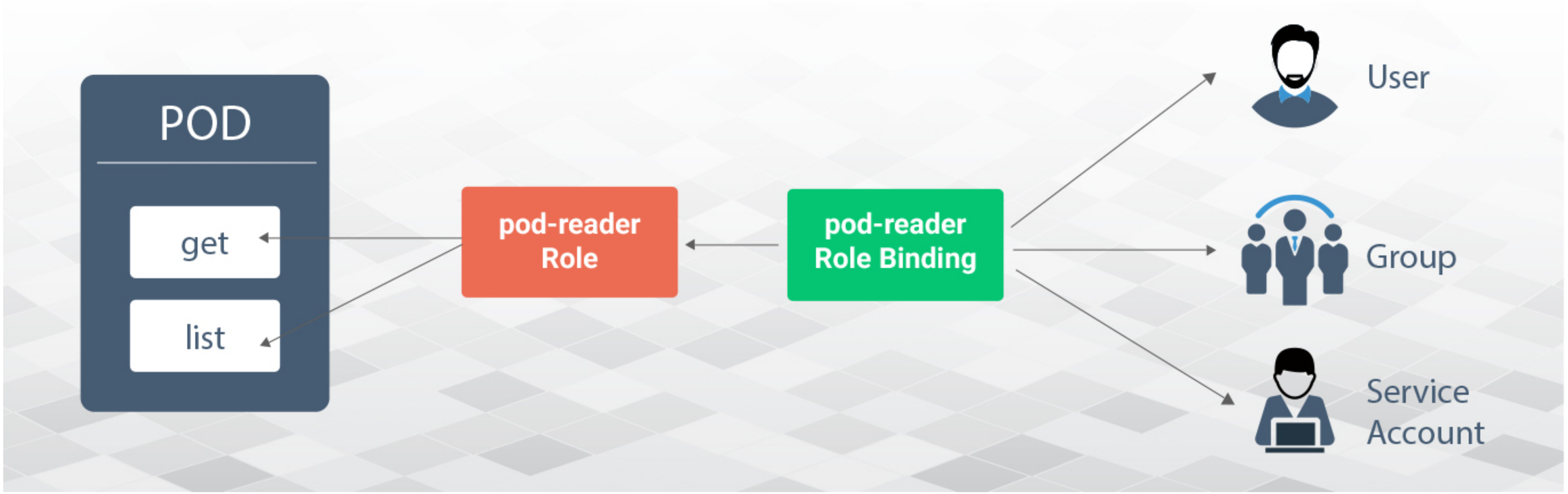
Что надо было сделать

- Унифицированный аутентификатор
- Управление пользователями и правами доступа между кластерами
- Систему защиты инфраструктуры от несанкционированного доступа

Централизованная аутентификация



RBAC Авторизация в Kubernetes

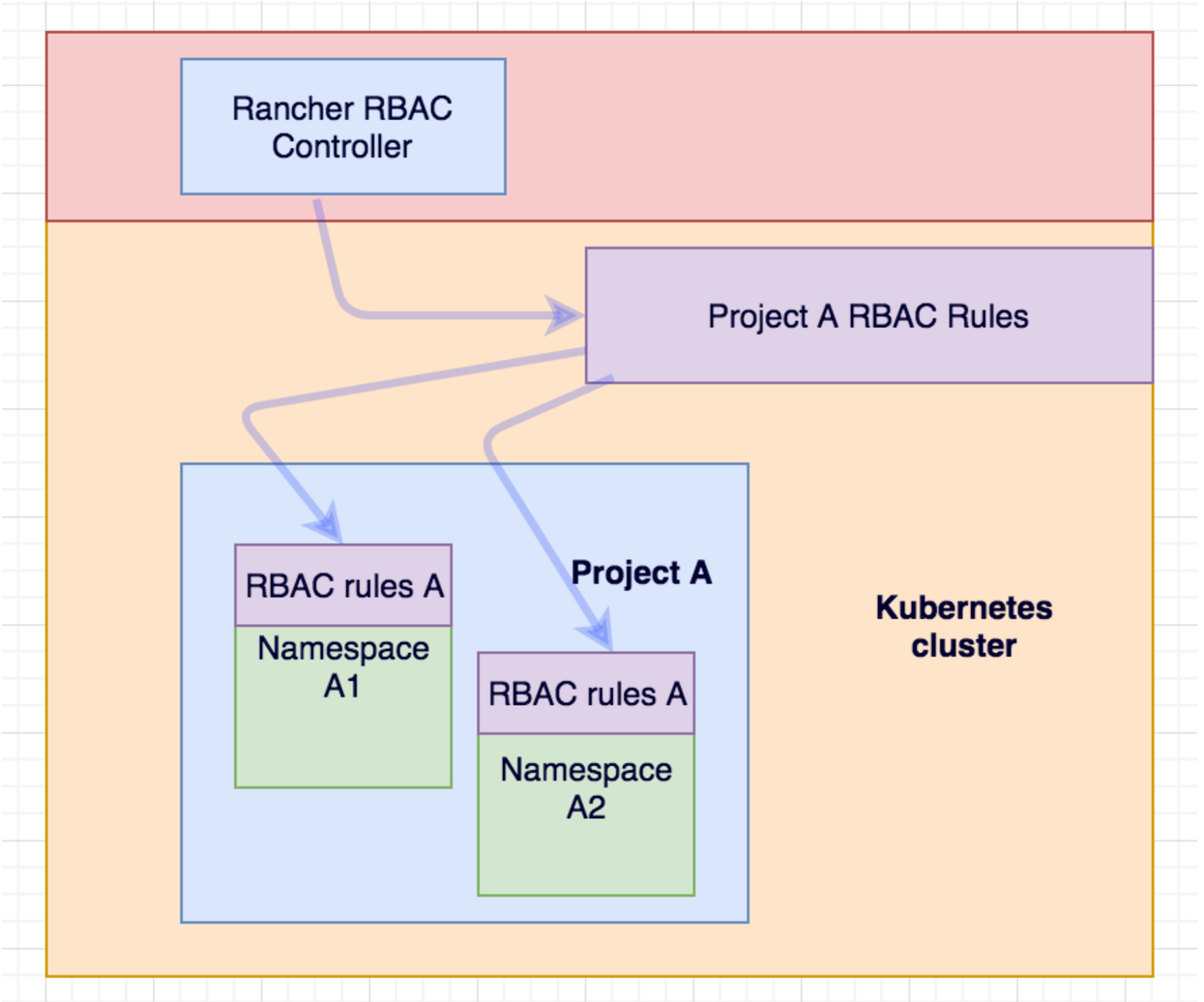


RBAC Авторизация в Rancher

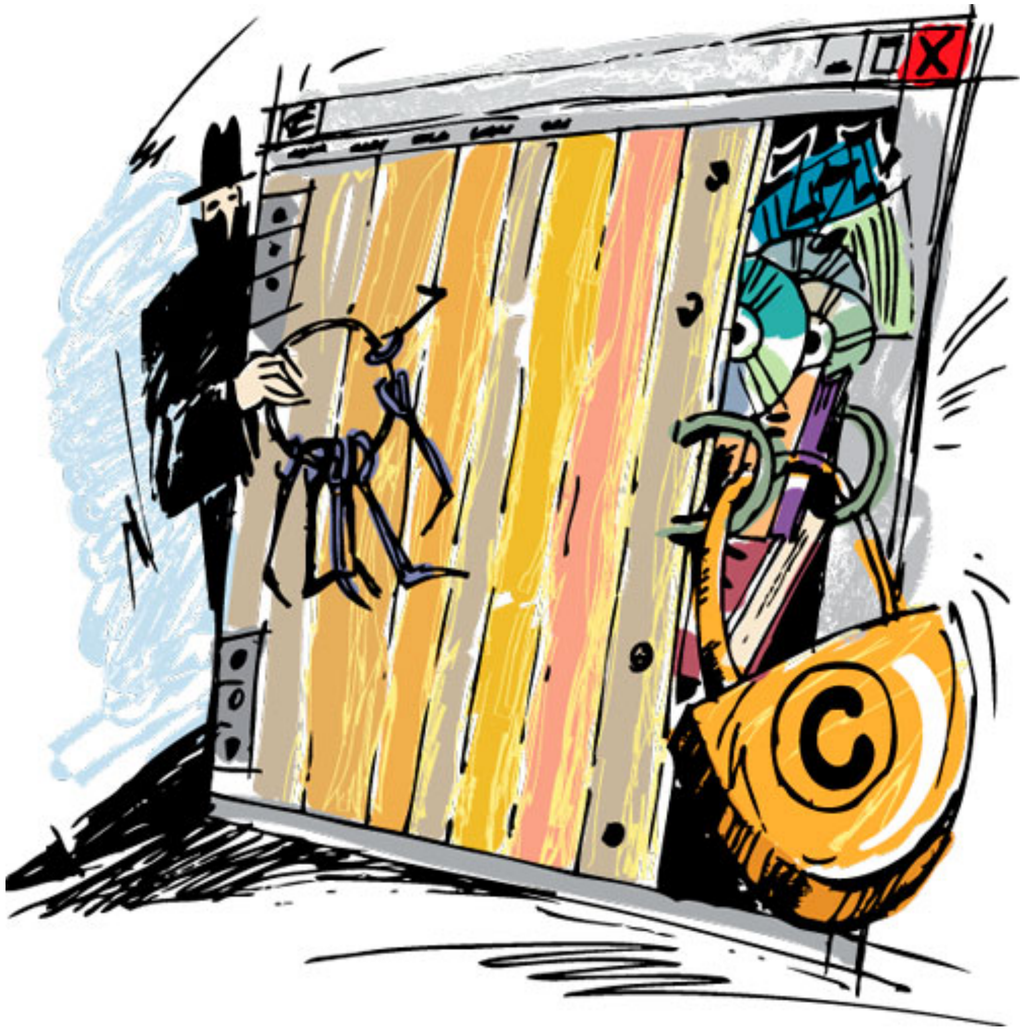
- Расширяет, а не заменяет Kubernetes RBAC
- Вводит концепцию “Проекта” - способа группировки Namespaces
- RBAC роли на проект
- Реализует автоматическое наследование прав доступа пользователем как только он добавлен в проект

Пример одного из Rancher RBAC controller

- Контроллер подписывается на события добавления и удаления пользователя в проект
- Копирует права доступа пользователя во все Namespaces проекта
- Удаляет права доступа по удалению пользователя



Защита инфраструктуры от “своих” так же важна, как защита на уровне API от “чужих”



Kubernetes Pod Security policy

- Предотвращает запуск контейнеров с root привилегией
- Ограничивает доступ контейнеров к определенным дискам
- Запрещает контейнеру открывать порты на хосте

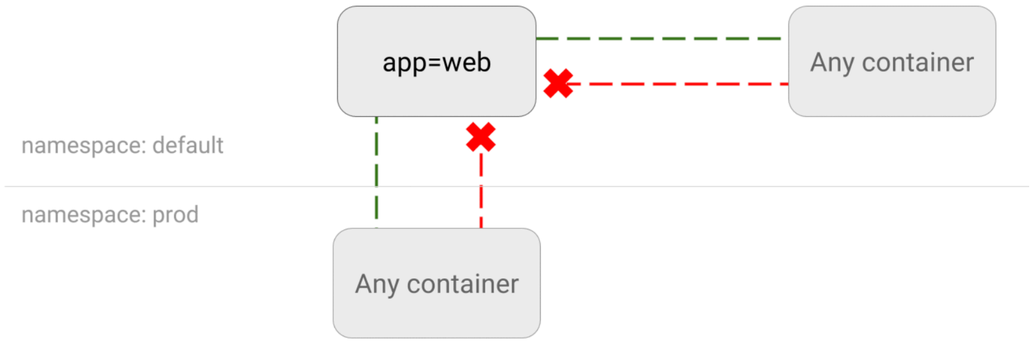
Pod Security Policy Template в Rancher

- Могут быть сконфигурированы единовременно
- Применены к целому кластеру, либо к индивидуальному проекту
- Наследование всеми контейнерами в кластере или проекте

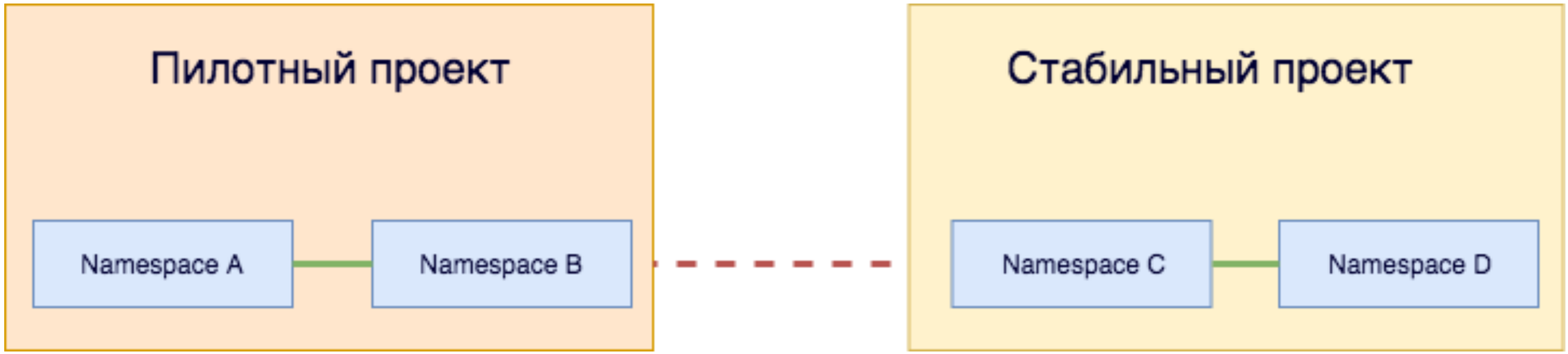
Kubernetes network policies

- По умолчанию все контейнеры имеют доступ друг к другу
- В идеале, администратор хотел бы регулировать и контролировать права доступа
- Network policy описывают правила доступа

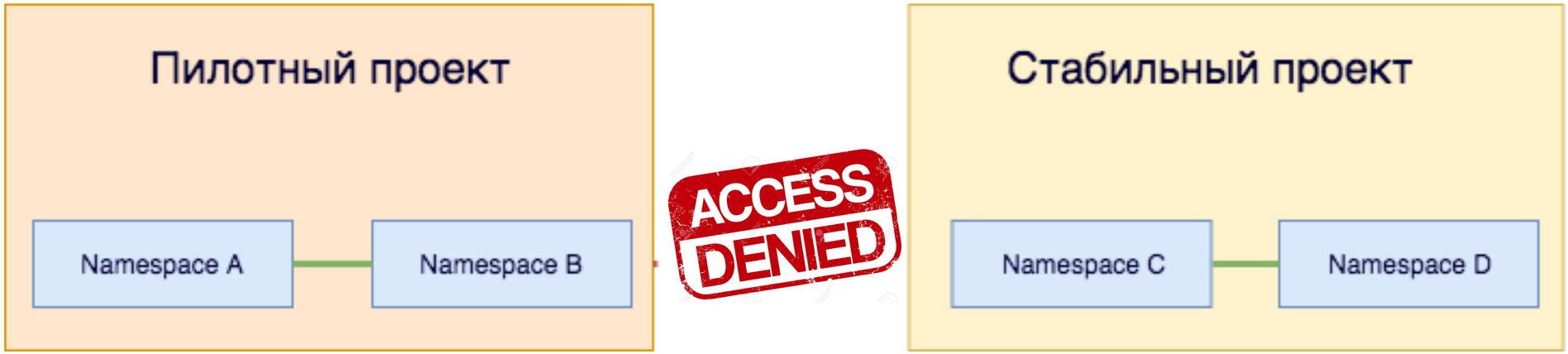
```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-deny-all
spec:
  podSelector:
    matchLabels:
      app: web
  ingress: []
```



Rancher расширяет network policies для поддержки multitenancy



Rancher расширяет network policies для поддержки multitenancy



Работа над ошибками

- Расширять, а не заменять, ключевую функциональность системы, с которой происходит интеграция
- Шаблоны часто используемых конфигураций как помощь администратору

Проблема #3 - Управление Kubernetes аддонами

Kubernetes аддоны

- Расширяют функциональность Kubernetes.
- Запускаются в контейнерах в самом кластере
- Апгрейд аддона выполняется независимо

Аддоны в облачных провайдерах Kubernetes не гибкие

- Предустановлены
- Не рекомендуется модифицировать
- Непрозрачные апгрейды

Что надо было сделать

- Установить важные аддоны по умолчанию
- Предоставить кастомизацию как альтернативу
- Четко и унифицировано показать системные сервисы запущенные в кластере

Аддоны в RKE кластерах

- Конфигурируемы
- Можно поменять после установки
- Можно отключить, и запустить свой кастомный аддон

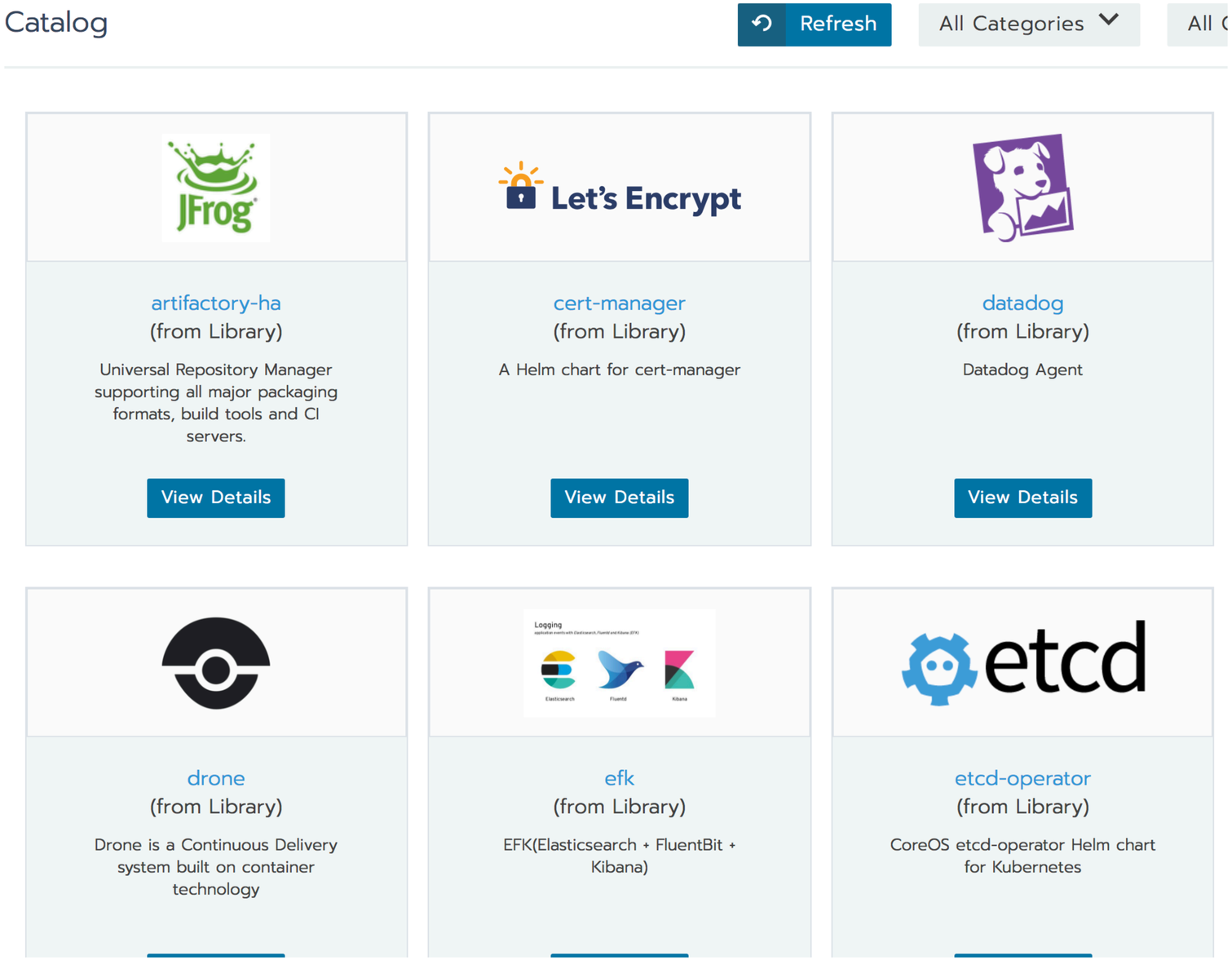
```
services:
  etcd:
    extra_args:
      election-timeout: "5000"
      heartbeat-interval: "500"
    snapshot: false
  kube_api:
    pod_security_policy: false
  kubelet:
    fail_swap_on: false
ssh_agent_auth: false

ingress:
  provider: "nginx"
kubernetes_version: "v1.10.3-rancher2-1"
network:
  plugin: "calico"
  calico_network_provider:
          cloud_provider: aws

addon_job_timeout: 30
authentication:
  strategy: "x509"
bastion_host:
  ssh_agent_auth: false
ignore_docker_version: true

```

Кастомные аддоны через каталог



Работа над ошибками

- Для простых кейсов хороша установка аддонов по умолчанию, а для сложных необходима поддержка кастомизации
- Прозрачность системных сервисов

наше мотто

- Нет привязки к конкретному облачному провайдеру
- Отсутствие привязки к Rancher API/UI
- Расширение, а не замена функциональности Kubernetes

Demo time

Спасибо!

Alena Prokharchyk,

Principal Software Engineer @RancherLabs

 @lemonjet

 alena1108