Kubernetes. Модель безопасности и контроллеры задач

План

- Модель безопасности Kubernetes
- Контроллеры задач в Kubernetes

Модель безопасности Kubernetes

Namespaces

Namespaces

- Можно создать несколько виртуальных кластеров в рамках одного физического кластера.
- **Namespace** - один виртуальный кластер

Namespaces

- Namespaces часто применяются для:
  - Обеспечения **multitenancy** (множественная аренда)
  - разграничения прав между командами
  - делегирования части административных функций доверенным пользователям
  - лимитирования ресурсов на проект с помощью квот (cpu, memory, storage)

Namespaces

- Достигается это путем:
  - Создания области видимости имен (Names)
    - Имя (Name) - минимальный объект Kubernetes API (/apis/v1/namespaces/<namespace>/pods/<name>)
  - Подключения политик безопасности и авторизации к выделенной части кластера

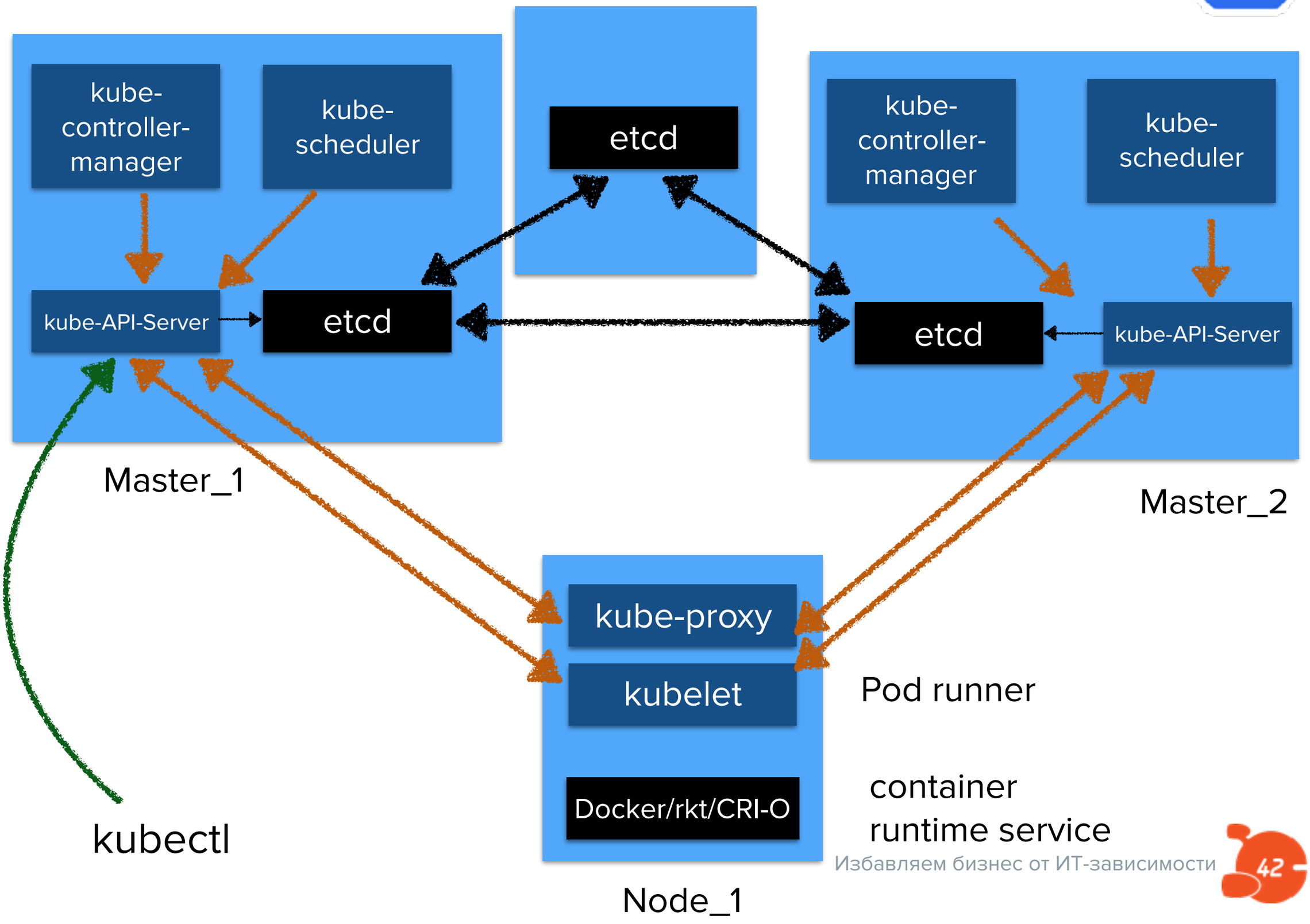
Namespaces

```
$ cat dev-1-namespace.yml
 ---
  apiVersion: v1
   kind: Namespace
   metadata:
    name: "dev-1"
```

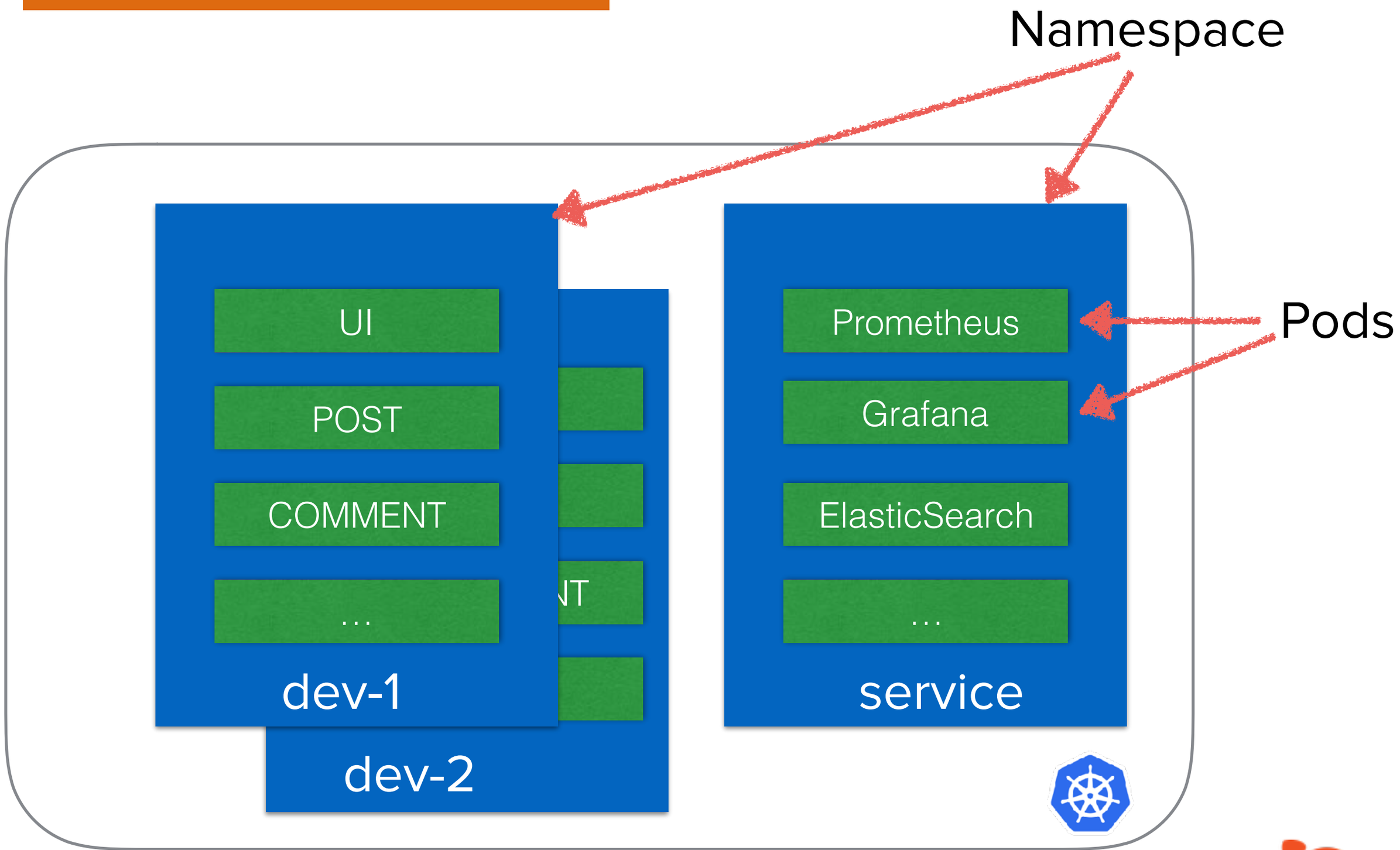
Namespaces

```
$ kubectl create ns dev-1
namespace "dev-1" created


$ kubectl create ns dev-1
namespace “dev-2" created


$ kubectl create ns dev-1
namespace "services" created
```

kube cluster



Namespaces


Namespace
Pods
UI
POST
COMMENT
…
dev-1
dev-2
Prometheus
Grafana
ElasticSearch
…
service

Namespaces

Показать поды только в namespace dev-1

```
$ kubectl get pods -n dev-1
```

```
$ kubectl get pods --namespace dev-1
```

Namespaces

- Namespace'ы по умолчанию:
  - **default** - для объектов для которых не определен другой Namespace
  - **kube-system** - для объектов созданных Kubernetes'ом
  - **kube-public** - для объектов к которым нужен доступ из любой точки кластера

Namespaces

Что **НЕЛЬЗЯ** увидеть в Namespace:

- сами Namespace
- nodes
- PersistentVolumes
- non-resource объекты (Ex: /healthz)

Namespaces

- У каждой команды есть возможность независимо от других использовать кластер

- При этом у команды есть свои
  - ресурсы (pod'ы, service'ы и т.д.)
  - политики (кому и какие действия можно совершать на кластере)
  - Ограничения и квоты на ресурсы

Namespaces

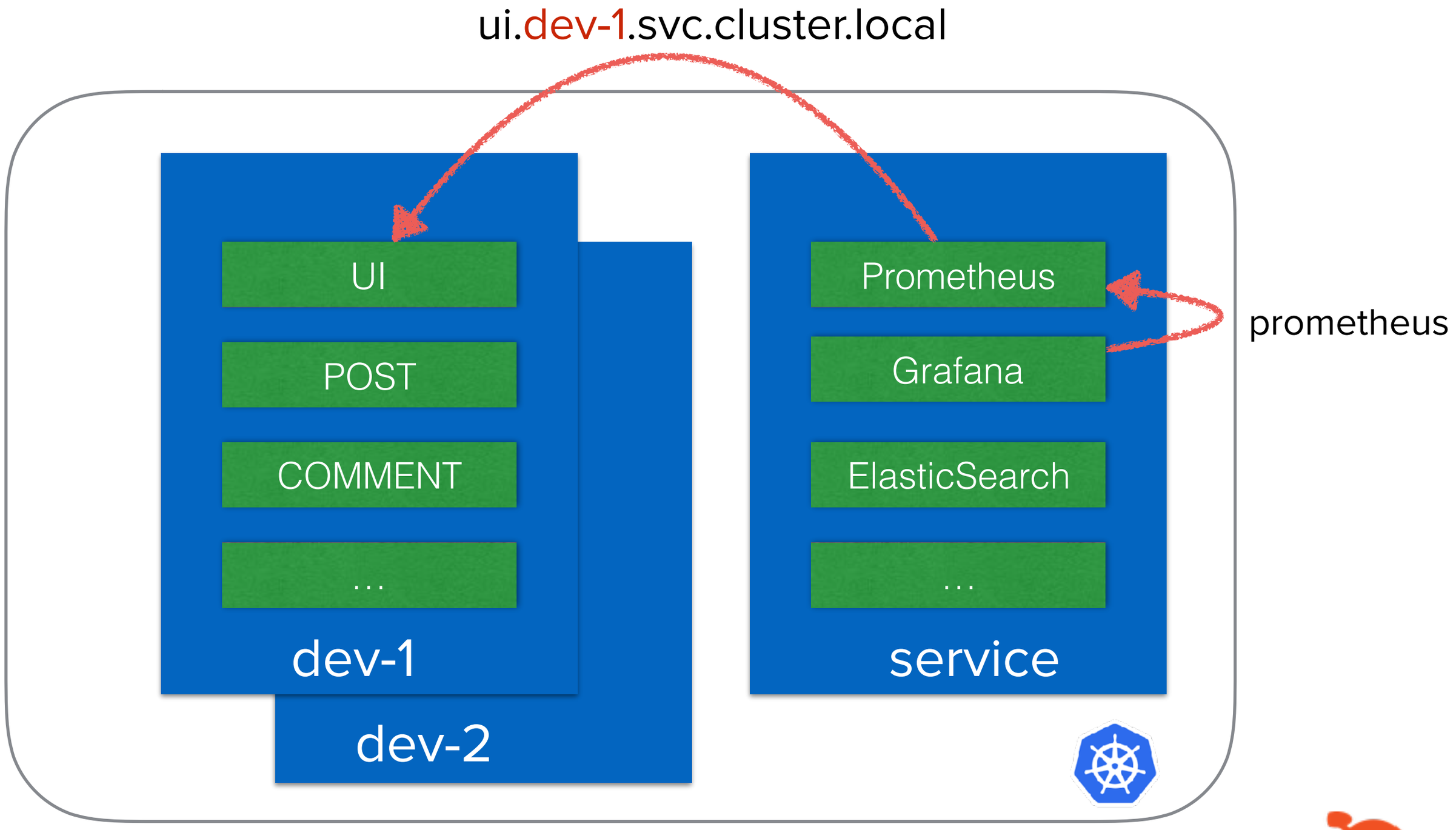
- Получить доступ к Service'у запущенному внутри Namespace'а можно по адресам:

  - <service-name>.<namespace>.svc.cluster.local
    запись создается автоматически

  - <service-name>
    внутри одного Namespace'а

  - FQDN
    Для доступа к сервису из множества Namespace'ов

Namespaces

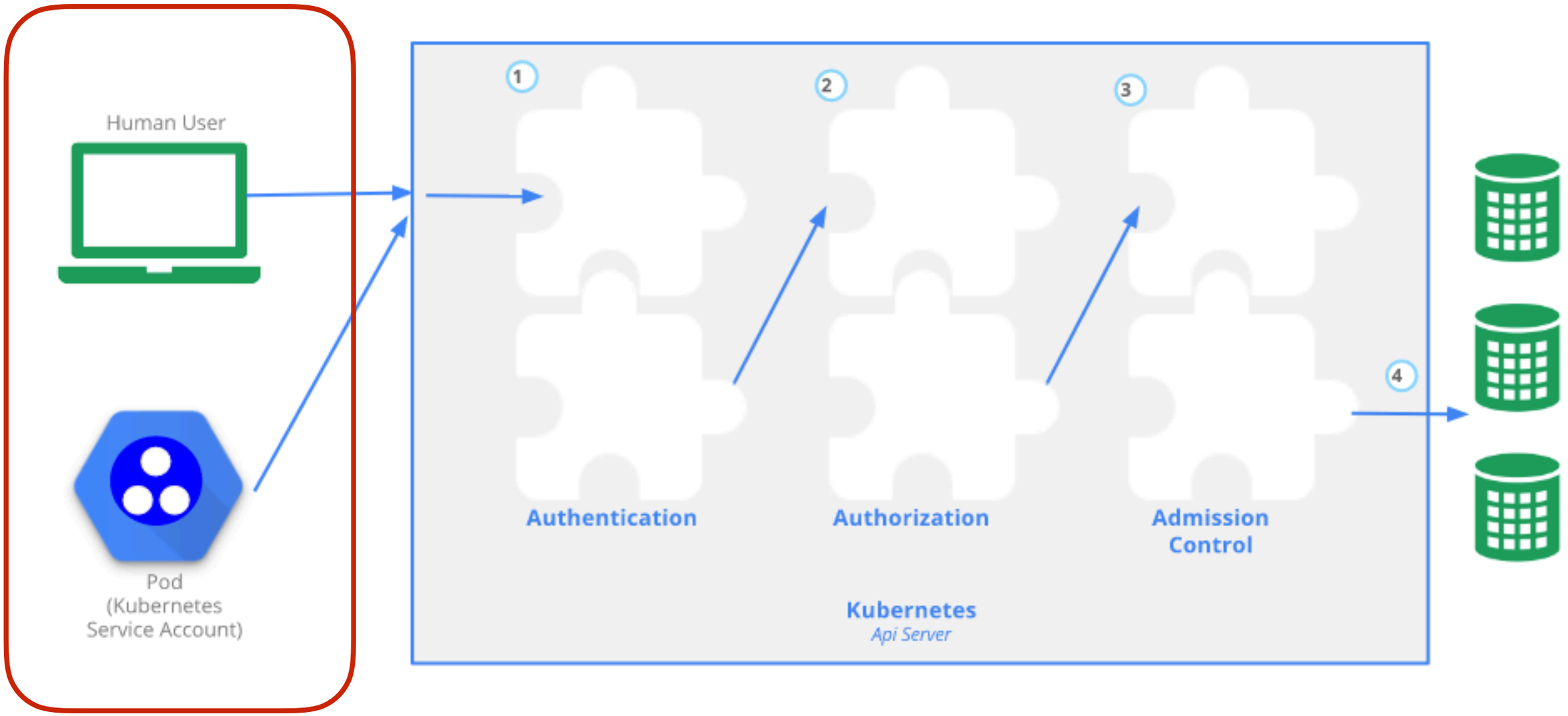

ui.dev-1.svc.cluster.local
UI
POST
COMMENT
...
dev-1
dev-2
Prometheus
Grafana
ElasticSearch
...
service
prometheus

Identity management

Accounts

- **User account** - учетная запись под которой работает пользователь
- **Service account** - учетная запись под которой процесс внутри pod'a работает с Kubernetes (например, с API)

Accounts


Human User
Pod
(Kubernetes
Service Account)
1
2
3
4
Authentication
Authorization
Admission
Control
Kubernetes
Api Server

Service accounts vs User accounts

Service Accounts

- Для задач в Pod-ах
- разделены по Namespace’ам
- Управляются Kubernetes
- Конфигурация приложения могут включать Service Account

User Accounts

- Для людей
- Работают во всем кластере
- Управляются внешними сервисами

User accounts

- User Accounts не могут быть созданы через API Kubernetes (нет такого объекта в Kubernetes API)
- В качестве источников информации могут использоваться
  - Файлы (токены, сертификаты)
  - LDAP
  - SAML
  - Kerberos

Service accounts: Автоматизация

- Service Account Admission Controller
- Token Controller
- Service Account Controller

Service Account Admission Controller

- Если у Pod'а не установлен Service Account, то устанавливает Service Account "default"
- Проверяет, что Service Account Pod'а существует
- Если у Pod'а нет своих ImagePullSecrets, то устанавливает ImagePullSecrets Service Account'а
- Подключает к Pod'у volume с ключом для API **/var/run/secrets/kubernetes.io/serviceaccount**

Service Account

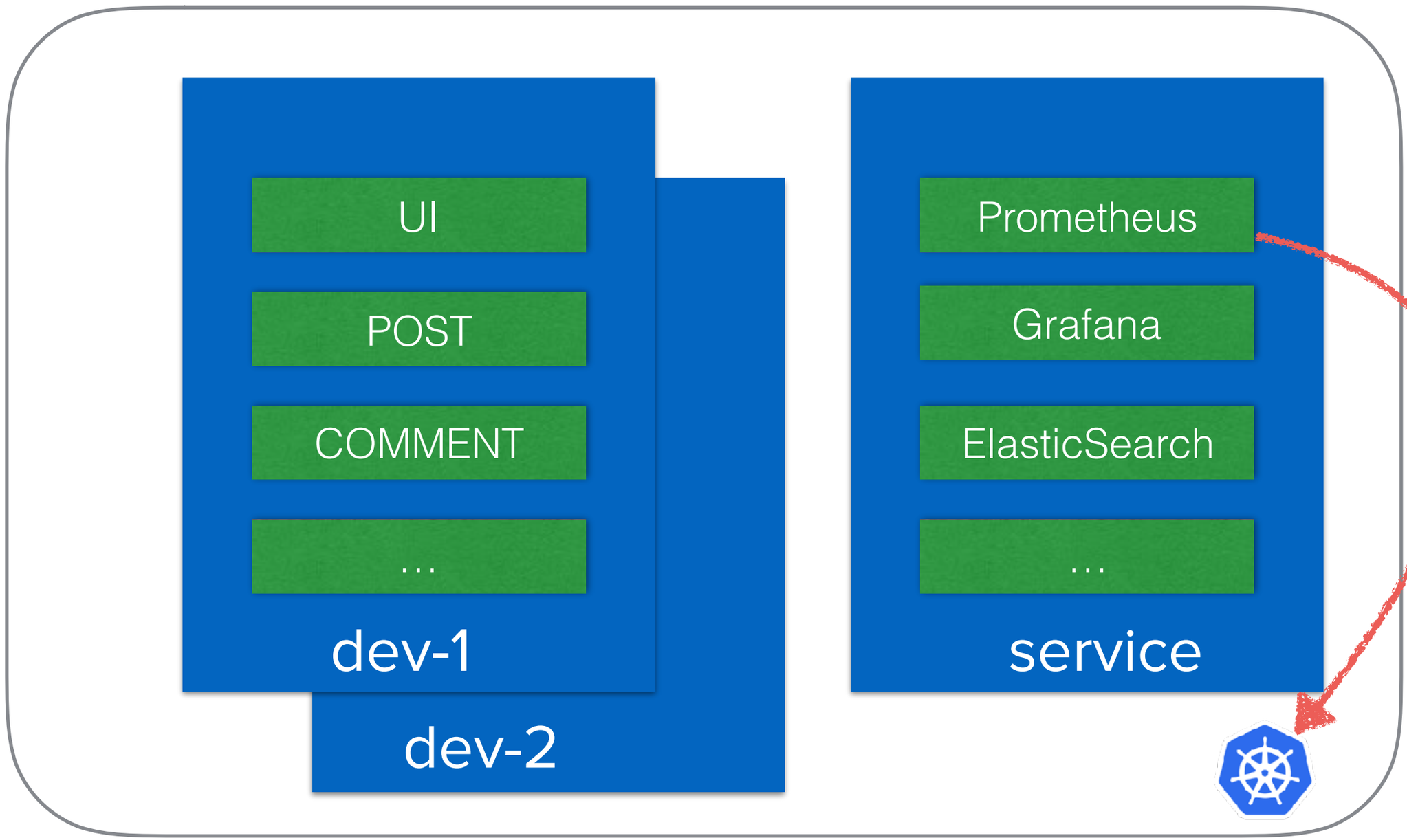
```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
  namespace: system-default
```

```
...
    - job_name: 'kubernetes-nodes'
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
```

Service Account

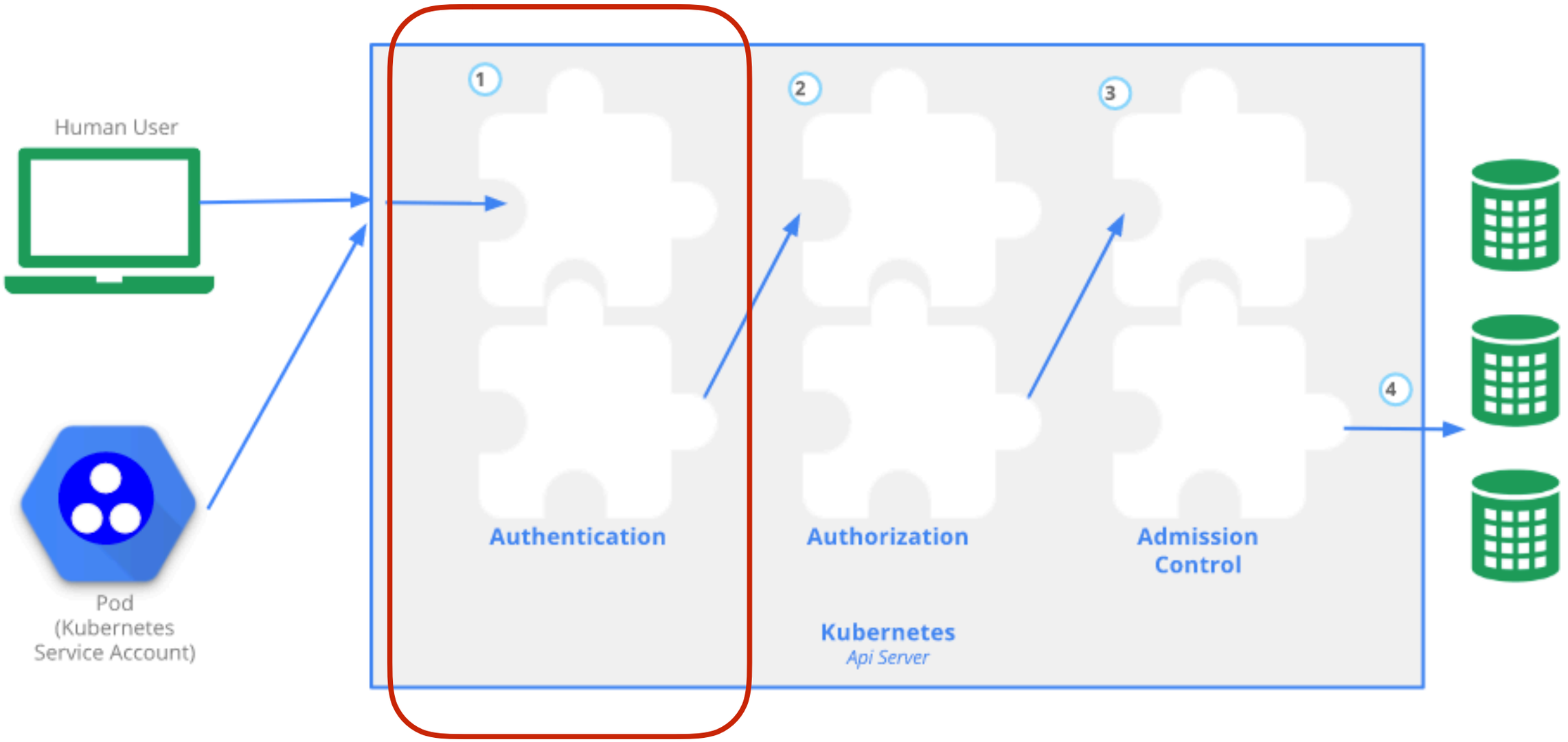
```
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: prometheus-core
  namespace: system-default
spec:
 replicas: 1
  template:
    metadata:
      name: prometheus-main
      labels:
        k8s-app: prometheus
        component: core
    spec:
     serviceAccountName: prometheus
…
```

Namespaces


UI
POST
COMMENT
...
dev-1
dev-2
Prometheus
Grafana
ElasticSearch
...
service

Аутентификация

Аутентификация



Аутентификация

- X509 сертификаты
- Статичные токены
- Bootstrap токены
- Статичные файлы с паролями
- Authenticate Proxy

Могут использоваться сразу несколько методов!

Аутентификация

- Основные поля:
- **Username** - идентификация конечного пользователя
- **UID** - уникальный идентификатор конечного пользователя
- **Groups** - объединение нескольких пользователей в 1 группу

x509

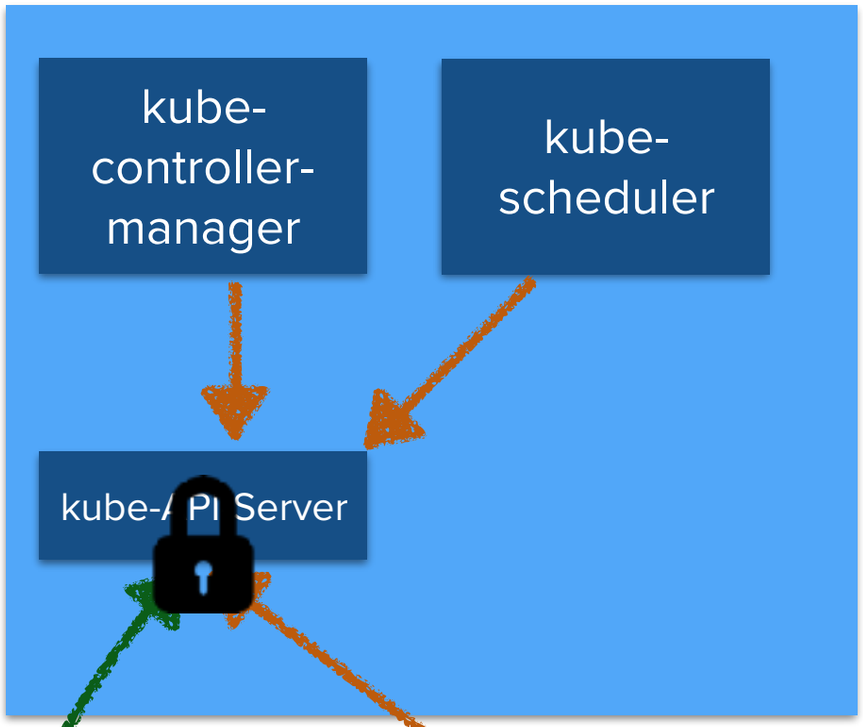
- сертификаты должны быть подписаны корневым сертификатом кластера

```
kube-apiserver --client-ca-file=ca.crt
```

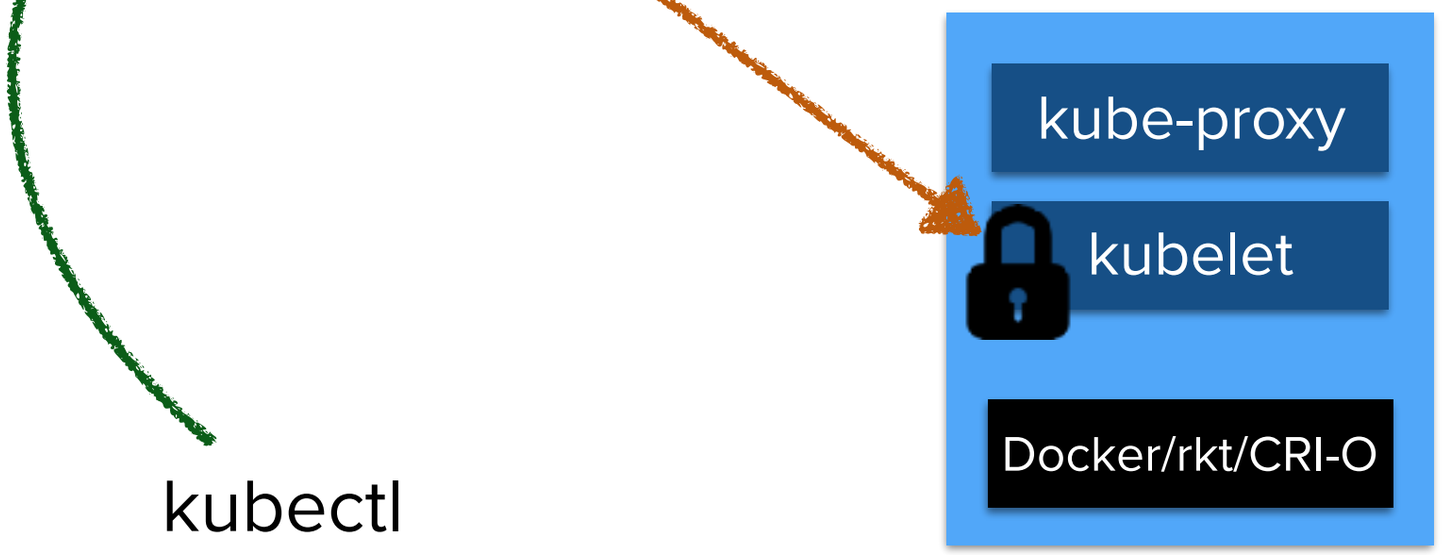
Создать запрос

```
openssl req -new -key jbeda.pem -out jbeda-csr.pem \
-subj "/CN=jbeda/O=app1/O=app2"
```

kube cluster



Master_1



Pod runner

container
runtime service

Node_1

kubectl

Tokens

- Статичные токены
    - лежат в папке (чаще всего прямо на master-ноде)

```
kube-apiserver --token-auth-file=SOMEFILE
```

```
$ cat SOMEFILE
token,user,uid,”group1,group2,group3"
…
31ada4fd-adec-460c-809a-9e56ceb75269,admin,42,"group1,group2,group3"
```

Token Controller

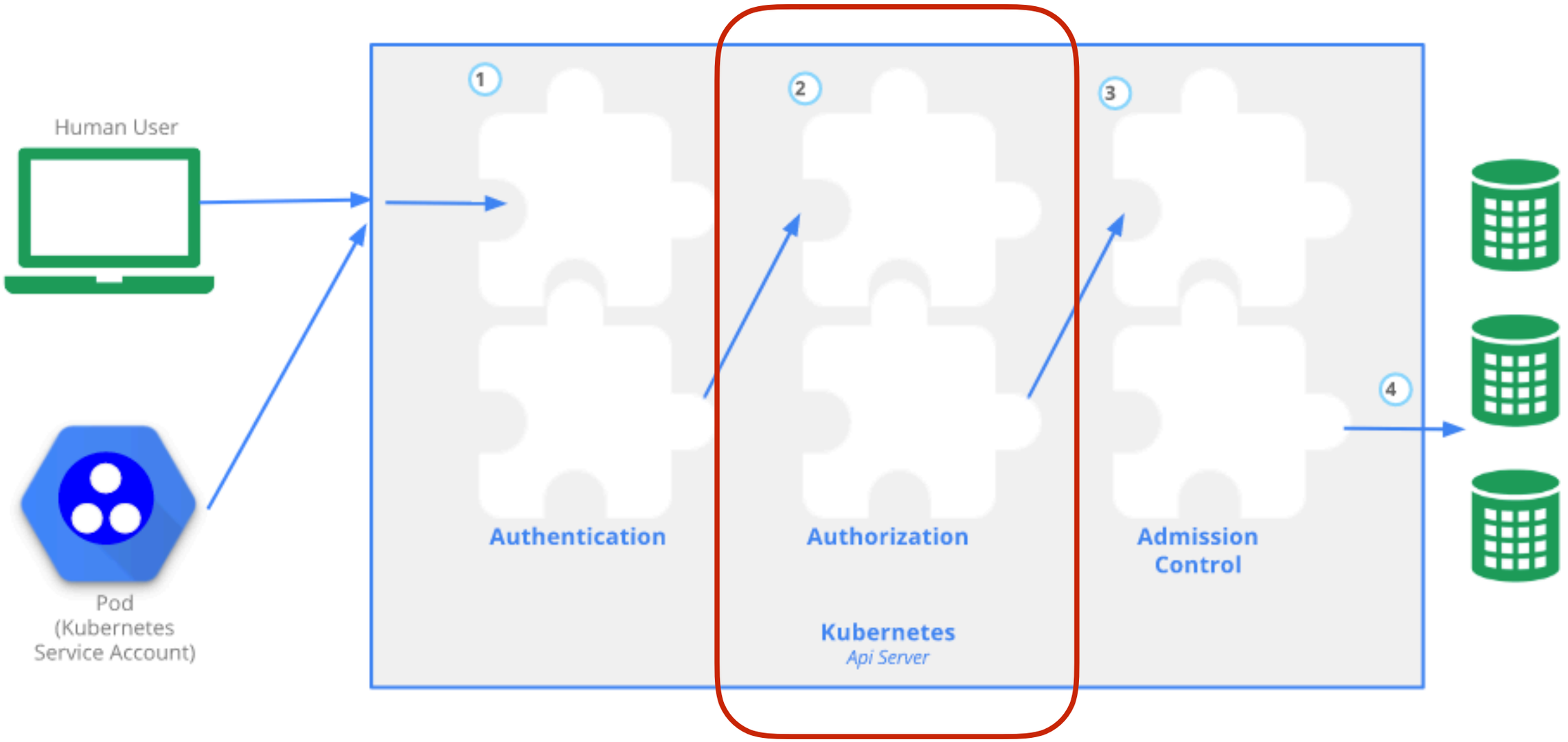
- Следит за созданием и удалением токенов для Service Account-ов

- Создает и удаляет соответствующие ключи доступа к API

- Следит, чтобы ключи всегда соответствовали существующему Service Account'y

Авторизация

Авторизация



Авторизация

- Attribute-based access control (ABAC)
- **Role-based access control (RBAC)**
- Node
- Webhook

ABAC

- Авторизирует действия пользователя на основе набора политик

- Политики состоят наборов атрибутов по которым происходит авторизация

- Виды атрибутов: пользовательские атрибуты, атрибуты ресурсов, объектов, окружений и т.д.

ABAC

- Сервер стартует с флагом --authorization-policy-file=Policy.json

Policy.json

Одна запись (атрибут)

```
{"apiVersion":
    "abac.authorization.kubernetes.io/v1beta1",
    "kind": "Policy",
    "spec": {
    "user": "alice",
    "namespace": "*",
    "resource": "*",
    "apiGroup": “*"}
}
```

RBAC

- Авторизация происходит на основании роли пользователя

- **Роль** - набор правил, задающих разрешения

- (ClusterRole - роль в рамках всего кластера

- Role - роль в рамках одного namespace

- Роли пользователям назначаются с помощью привязок (RoleBindings)

RBAC

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

Правило для чтения информации о pod-ax

1) Группа ресурсов
2) ресурсы
3) действия

RBAC

ClusterRole = Role + :

- кластерные ресурсы (nodes, PersistentVolumes)
- нересурсные эндпоинты (“/healthz”)
- ресурсы из нескольких или всех namespaces

RBAC

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups: ["extensions"]
  resources:
  - deployments
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
```

блок правил

RBAC-правила

```
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
verbs: ["get", "list", "watch"]
```

блок ресурсов

RBAC-правила

```
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
verbs: ["get", "list", "watch"]
```

блок разрешенных действий

RBAC-правила

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups: ["extensions"]
  resources:
  - deployments
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
```

RoleBinding

Назначаем роль:

```
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: services
```

RoleBinding

Назначаем роль:

```
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: services
```

Что привязываем?

RoleBinding

Назначаем роль:

```
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: services
```

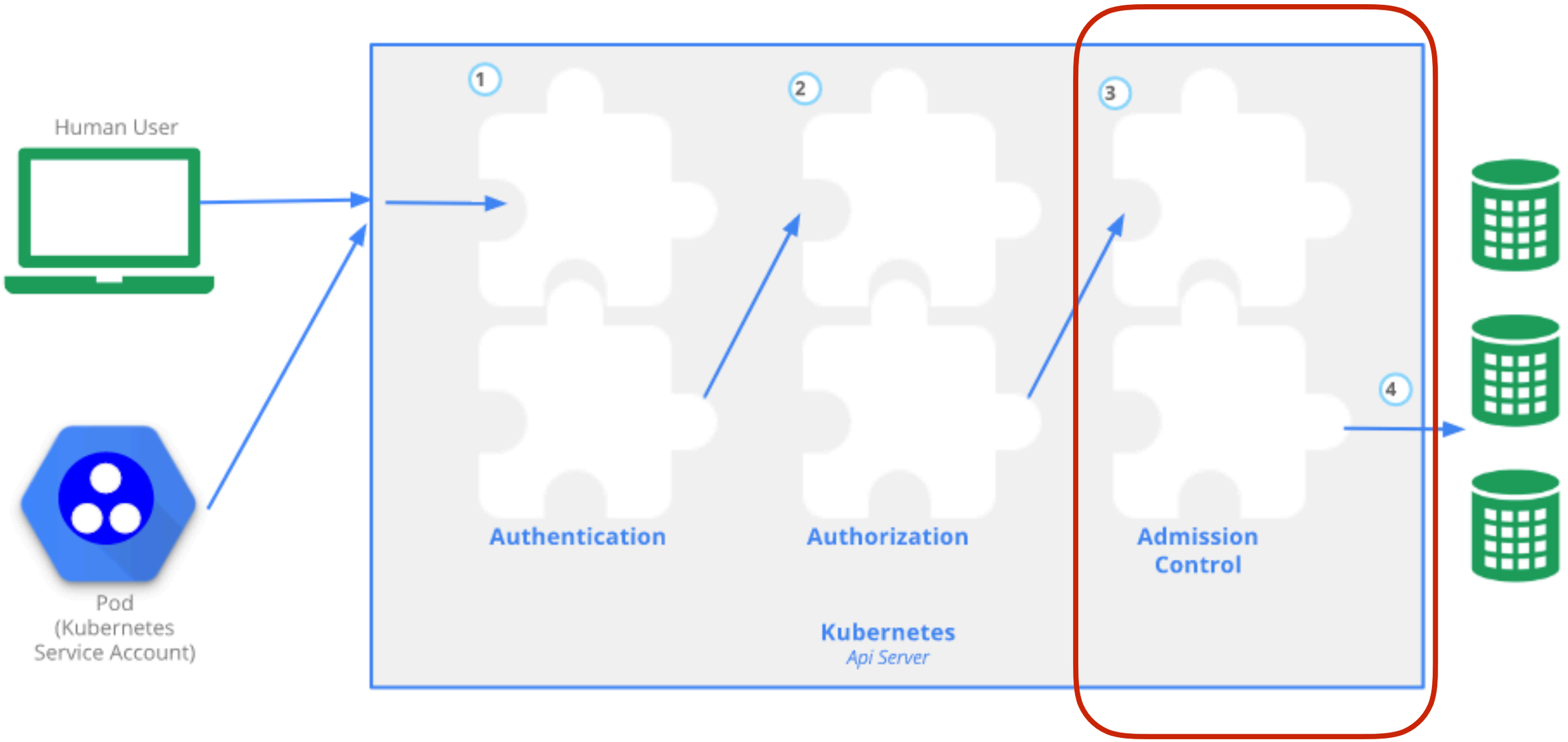
Кому привязываем?

RoleBinding

Назначаем роль:

```
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: services
- kind: User
  name: John
  apiGroup: rbac.authorization.k8s.io
- kind: Group
  name: admins
  apiGroup: rbac.authorization.k8s.io
```

Admission control



Admission controllers

- Набор модулей
  - AlwaysPullImages
  - NodeRestriction
  - ...
- Работает на основе действий (verbs) с ресурсами (отклоняет или разрешает)

Controllers

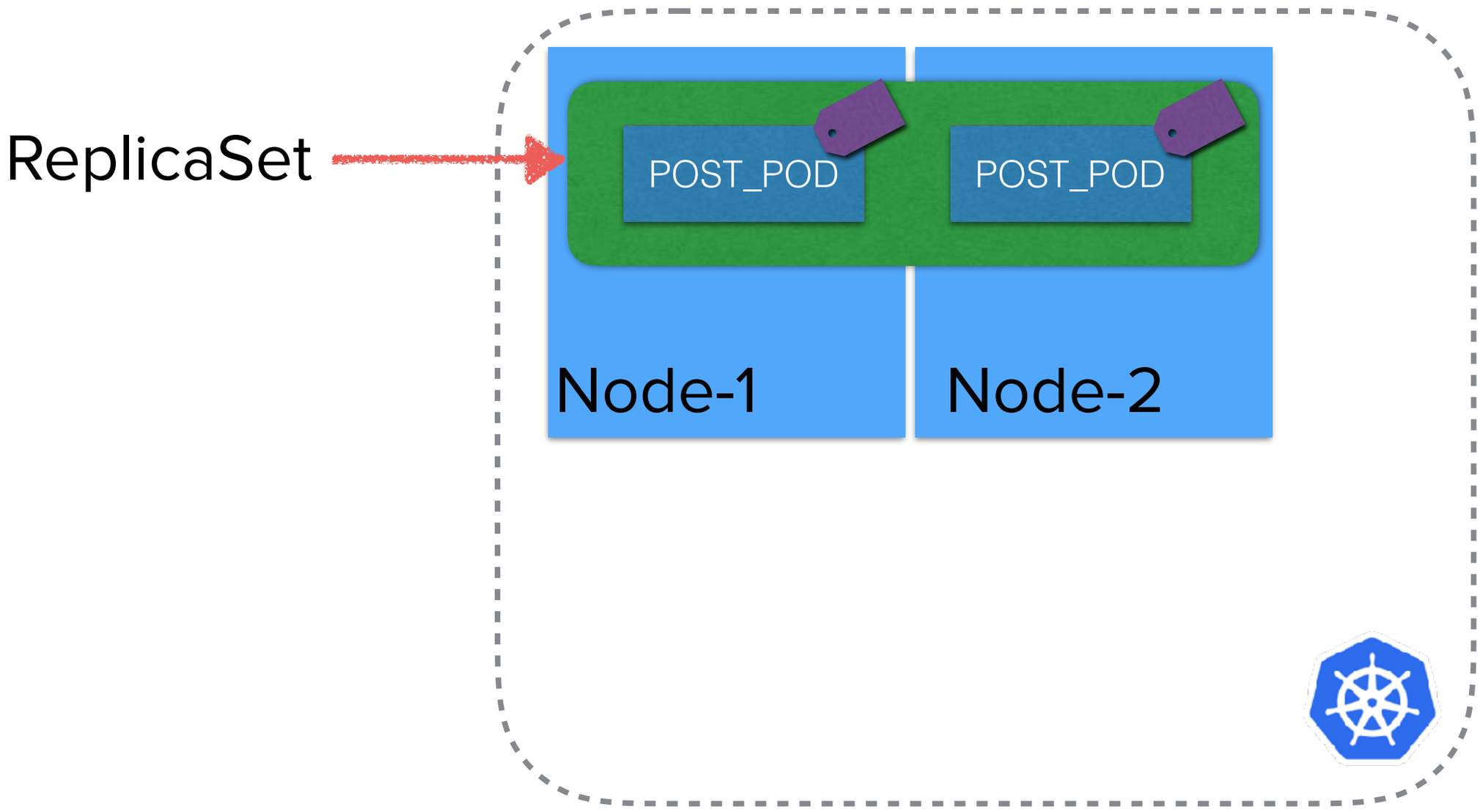
ReplicaSets

- Replica set обеспечивает, что в любой момент времени запущено необходимое количество Pod'ов
- ReplicationController + set-based label selector
- Не умеет “гладко” выкатывать новые поды

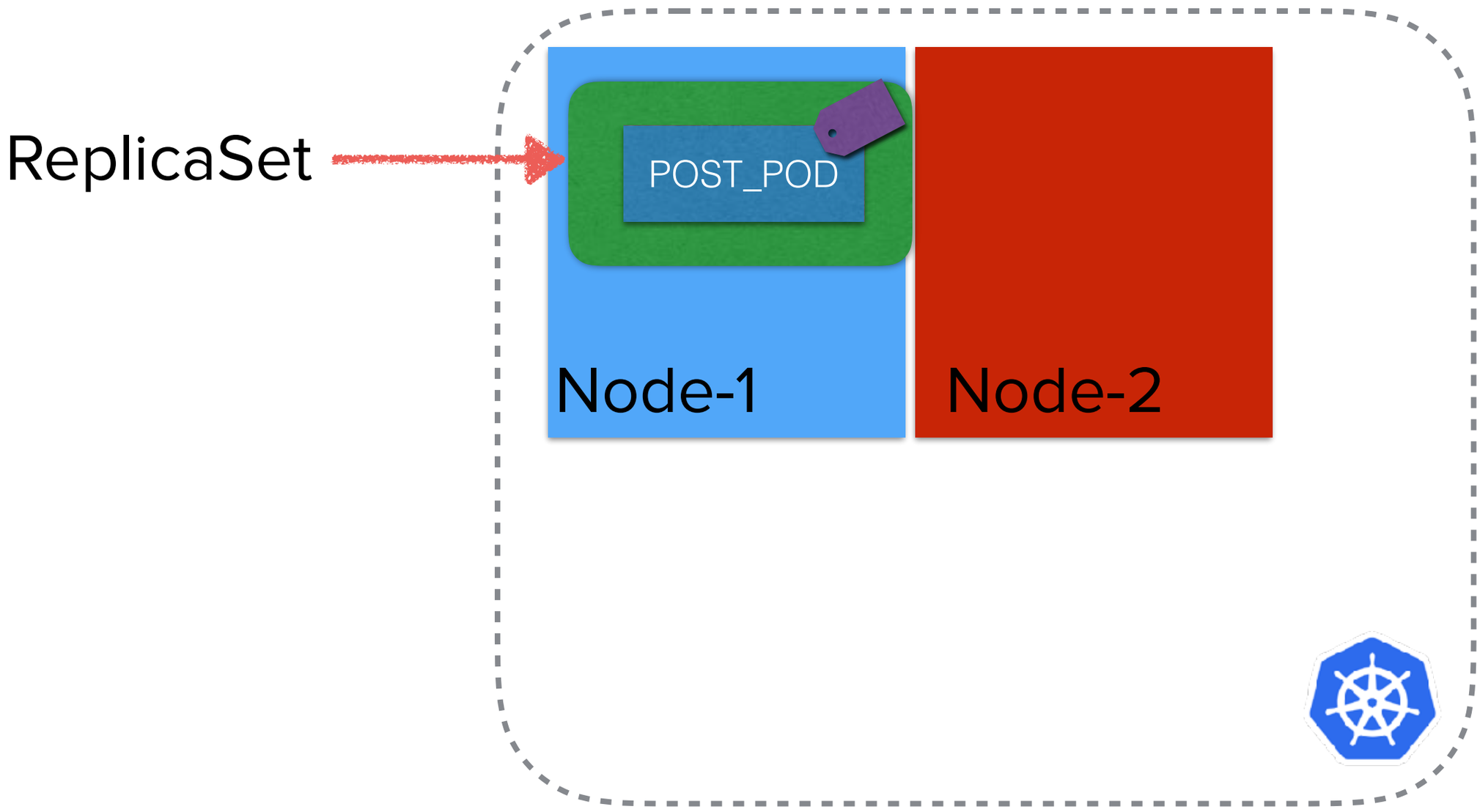
ReplicaSets

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: post
spec:
  replicas: 2
  selector:
    matchLabels:
      app: post
  template:
    metadata:
      name: post
      labels:
        app: post
    spec:
      containers:
      - image: chromko/post
        name: post
```

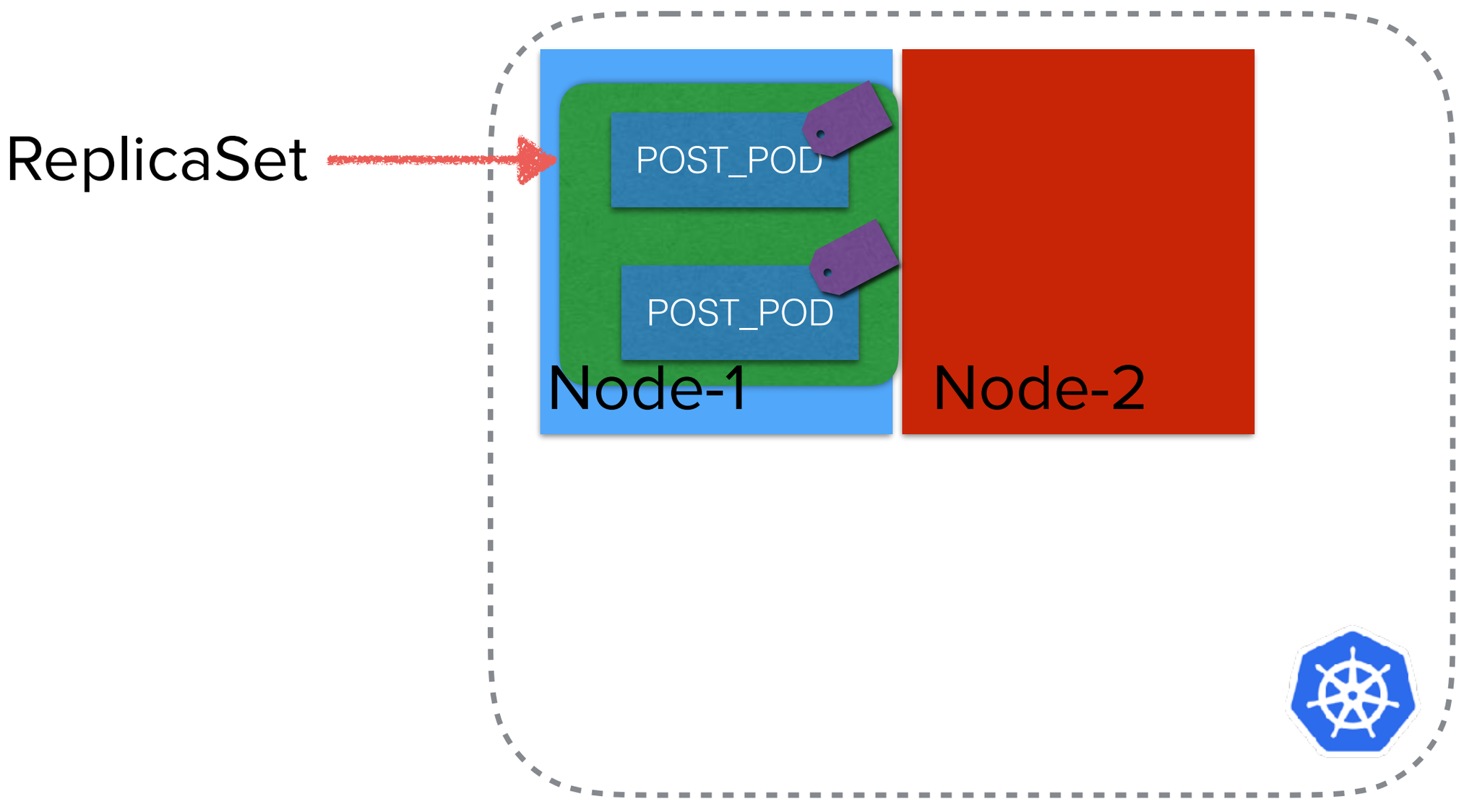
ReplicaSets



ReplicaSets

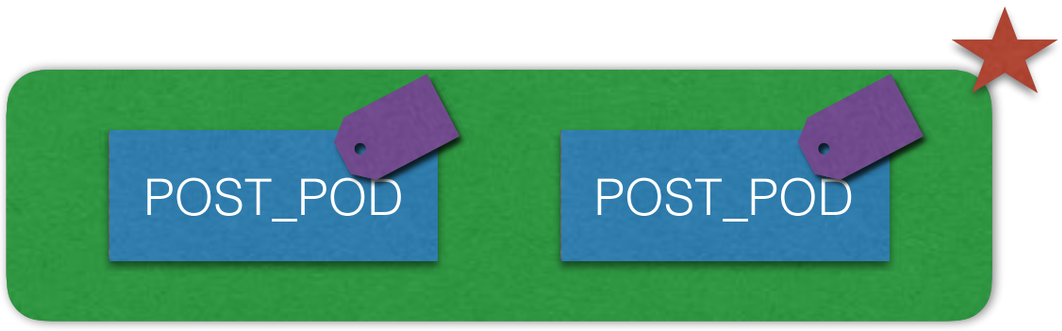


ReplicaSets

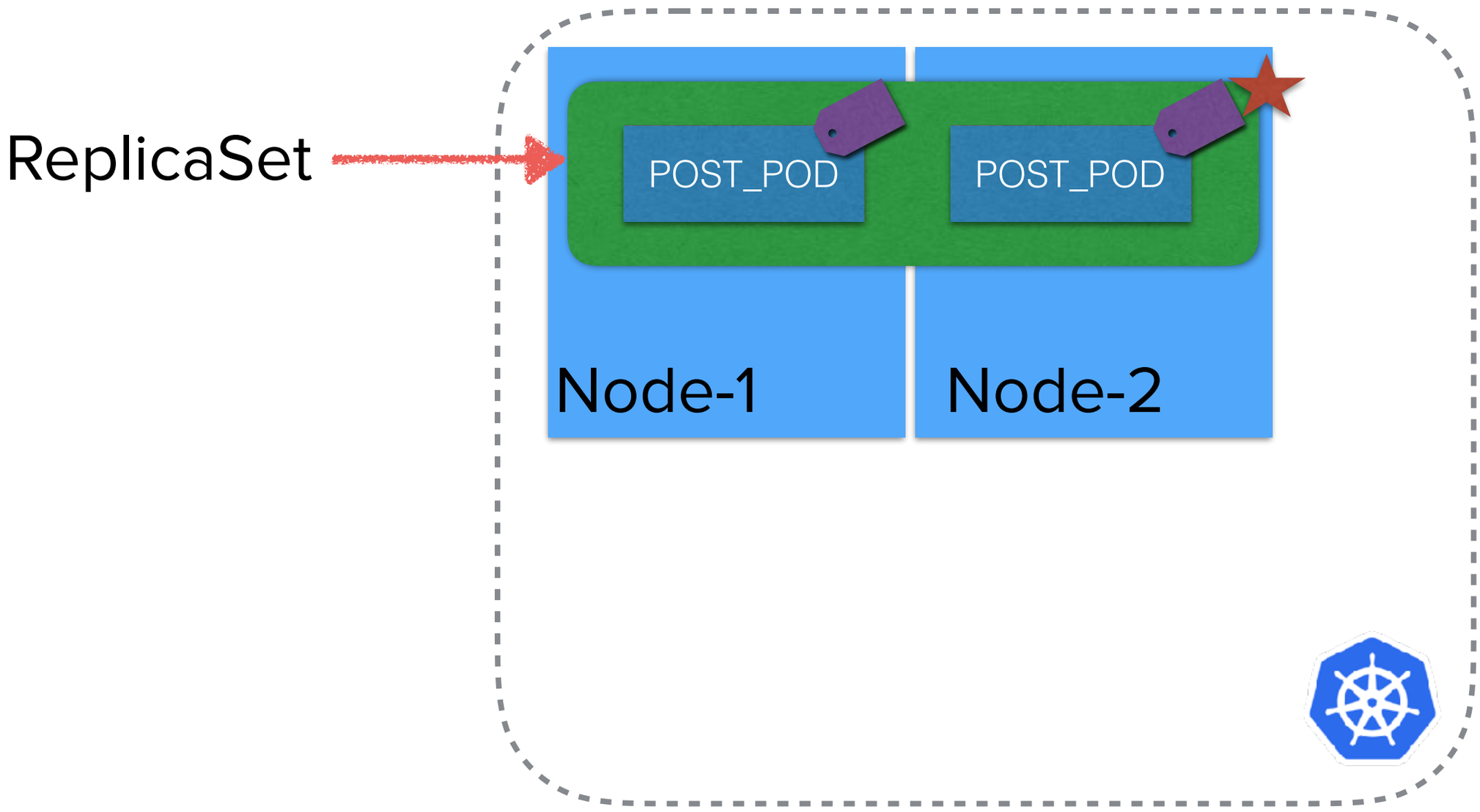


Deployments

- **Deployment** = ReplicaSets + ★ :
    - Текущее и желаемые состояния подов
    - Контроль процесса обновления состояния
    - Контроль процесса отката состояния



Deployments



StatefulSets

- Управляет выкаткой и масштабированием Pod’ов также как и Deployment

- Гарантирует очередность запуска и уникальность запущенных Pod’ов

- Гарантирует, что Pod не сменит свое расположение при смене состояния

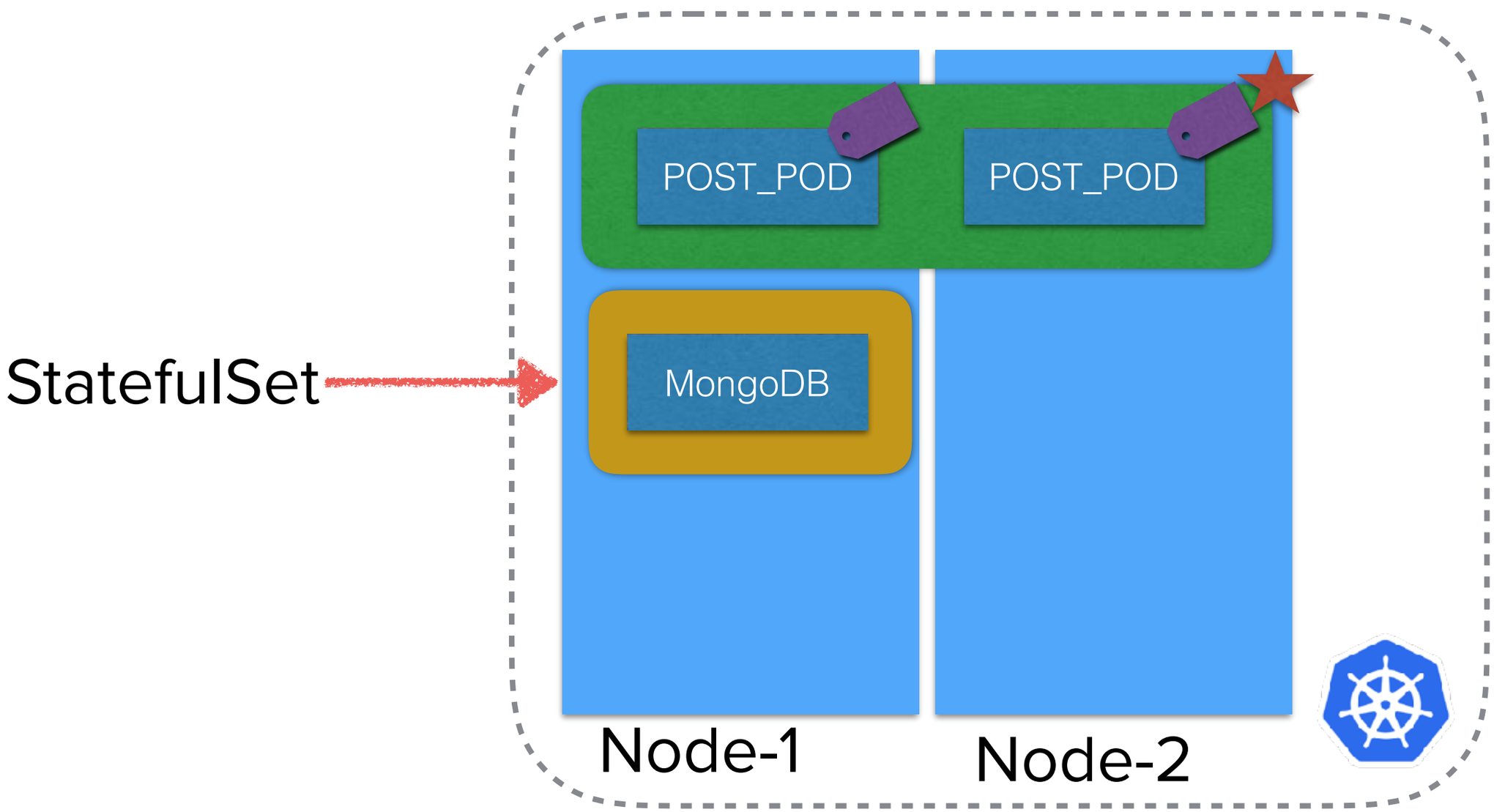
StatefulSets

- Применяется, когда приложению нужно:
  - Постоянные уникальные сетевые идентификаторы
  - Постоянное хранилище данных
  - Очередность запуска при выкатке и масштабировании
  - Очередность при выключении и удалении
  - Очередность при rolling updates

StatefulSets

- Ограничения
  - Находится в Бете
  - Хранилище должно быть преднастроено администратором или настроено с помощью PersistentVolume Provisioner'a
  - Удаление или масштабирование вниз не удалит ассоциированные с Pod'ами Volume'ы
  - Headless Service должен отвечать за сетевую идентификацию Pod'ов

ReplicaSets



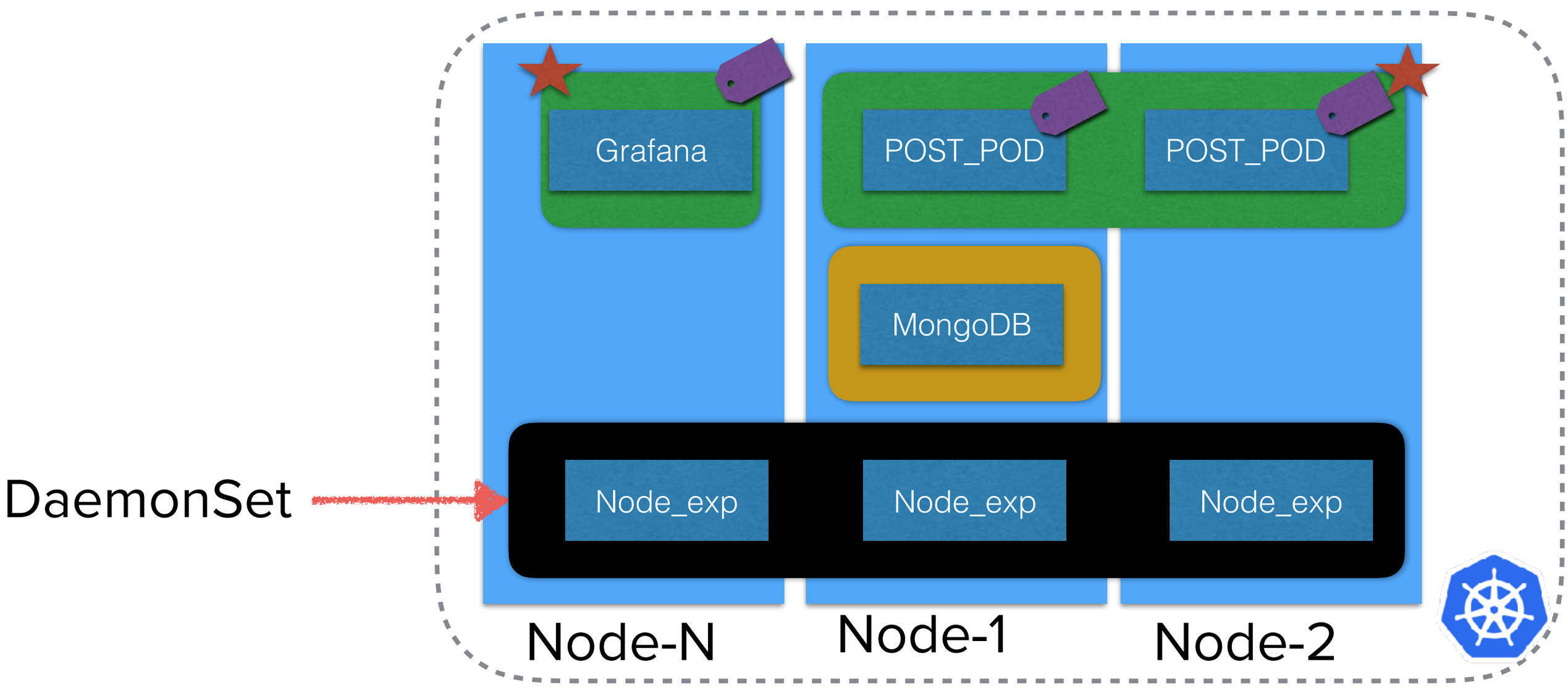
DaemonSet

- Запуск задачи на каждой ноде кластера
- DaemonSet-ы могут быть запущены даже пока Scheduler не стартовал
- Будут запущены еще до запуска остальных задач
- Поддерживается Rolling Update

DaemonSet

```
kind: DaemonSet
metadata:
  name: prometheus-node-exporter
  namespace: Service
  labels:
    k8s-app: prometheus
    component: node-exporter
spec:
template:
  metadata:
    name: prometheus-node-exporter
    labels:
      k8s-app: prometheus
      component: node-exporter
  spec:
    containers:
    - image: prom/node-exporter:0.12.0
      name: prometheus-node-exporter
      ports:
      - name: prom-node-exp
        containerPort: 9100
        hostPort: 9100
```

CronJobs


Grafana
POST_POD
POST_POD
MongoDB
DaemonSet
Node_exp
Node_exp
Node_exp
Node-N
Node-1
Node-2

Jobs

- Запускает 1 или несколько pod-ов
- Параллельный и последовательный запуски
- Ждет удачного завершения запущенных pod-ов

Cron Jobs

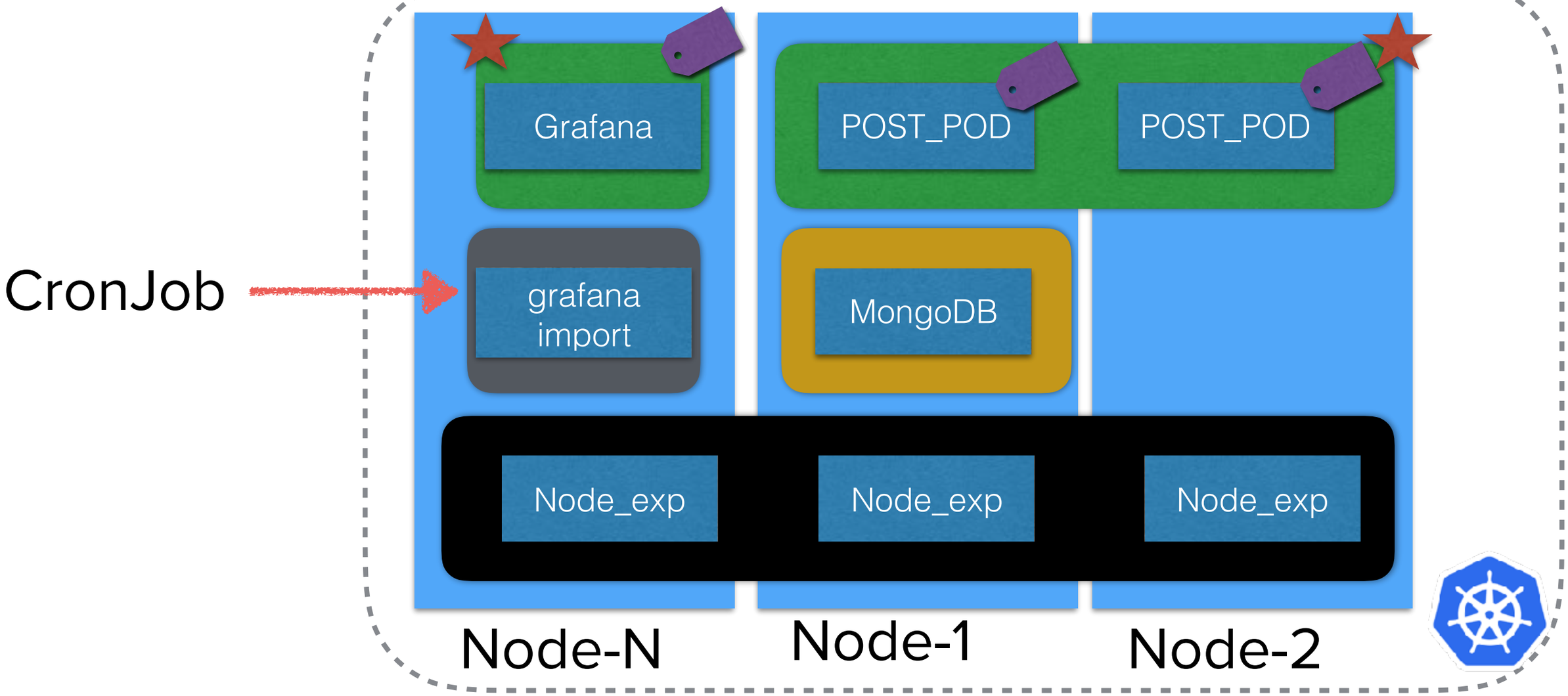
- Запуск Job-ов по расписанию в Cron
- единожды определенный момент времени
- периодически повторяя

Cron Jobs

```
---
apiVersion: batch/v2alpha1
kind: CronJob
metadata:
  name: grafana-import-job
  namespace: services
  labels:
    k8s-app: grafana
    component: import-dashboards
spec:
  schedule: "*/5 * * * *"
  jobTemplate:
   …

   curl --silent --fail --show-error \
 –request POST http://admin:admin@grafana:3000/api/dashboards/import
```

Каждые 5 минут

CronJobs


CronJob
Grafana
grafana import
POST_POD
POST_POD
MongoDB
Node_exp
Node_exp
Node_exp
Node-N
Node-1
Node-2

CronJobs

*Спустя 5 минут

CronJob