KARPOV.COURSES >>>
КОНСПЕКТ

# Конспект > 5 урок  > DE и Kubernetes, часть 2

## > Оглавление

# > Как хранить персистентные данные внутри Kubernetes

Хранилище внутри Pod'a эфемерно - это значит, что данные с запущенного приложения сохраняются в Pod'e и будут потеряны при его перезапуске.

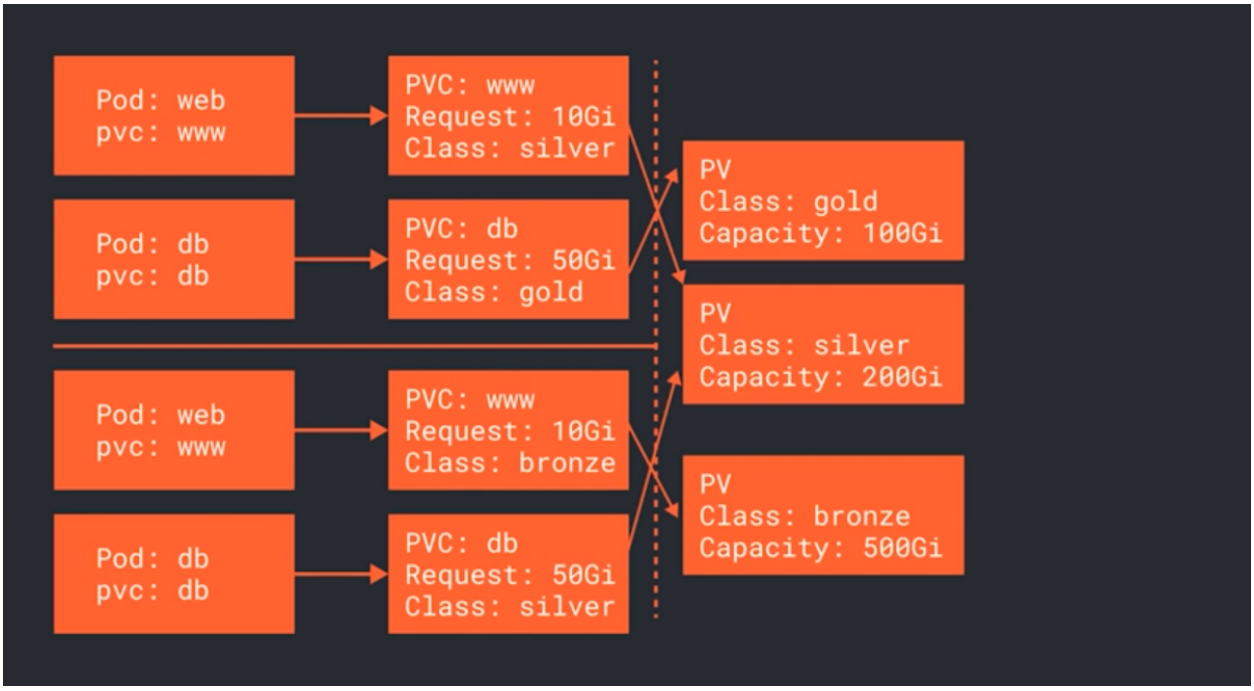
Чтобы не потерять данные, в Kuber'e для этого предназначены абстракции PersistentVolume и PersistentVolumeClaim.

В Pod'e у нас есть раздел volumes: список объектов volume, присоединенных к Pod и volumeMounts: список параметров монтирования.

```
apiVersion: v1
kind: Pod
metadata:
 name: volume-example
spec:
 containers:
 - name: nginx
   image: nginx-stable-alpine
   volumeMounts:
   - name: html
   mountsPath: /usr/share/nginx.html
   ReadOnly: true
 - name: content
   image:alpine.latest
   command: ["/bin/sh","-c"]
   args:
   - while true; do
      date >> /html/index.html;
      sleep 5;
     done
   volumeMounts:
   - name: html
   mountsPath: /html
  volumes:
  - name: html
  emptyDir: 0
```

Здесь показан механизм монтирования постоянных PersistentVolumes. В Pod'e мы указываем PersistentVolumeClaim (PCV) - запрос к системе на предоставления диска для хранения постоянных данных (которые не пропадут после перезапуска).

В разделе PCV (PersistentVolumeClaim) мы указываем его имя, необходимый нам объем памяти и какого класса (class) мы хотим получить диск.




На основе PersistentVolumeClaim ваш облачный провайдер выделит из файлового хранилища PVC необходимого объема и типа.

PersistentVolume создаются вот таким образом:

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: nfsserver
spec:
 capacity:
  storage: 50Gi
 volumeMode: Filesystem
 accessModes:
  - ReadWriteOnce
  - ReadWriteMany
 persistentVolumeReclaimPolicy: Delete
```

```
 storageClassNew: slow
 mountOptions:
  - hard
  - nfsvers=4.1
 nfs:
  path:/exports
  server: 172.22.0.42
```

Обратите внимание на:
- capacity.storage: общий объем запрашиваемого storage;
- accessModes: список поддерживаемых режимов подключения:
  - ReadWriteOnce
  - ReadWriteMany
  - ReadOnlyMany

В облаке очень часто вам не нужно создавать отдельно PersistentVolume, вы создаете только запрос - абстракцию PersistentVolumeClaim. На ее основе облако выделит и создаст PersistentVolume требуемого объема и типа.

Важный момент, мы задаем:

- persistentVolumeReclaimPolicy - это поведение PVC, которые были удалены;

- Retain - остается;

- Delete - удаляется методом storage provider'a

- storageClassName: имя класса storage.

Вот так выглядит PersistentVolumeClaim.

Мы задаем тип, указываем имя, указываем storageClassName и указали в каком режиме мы будем его монтировать - ReadWriteOnce.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: nginx-pvc-ssd
spec:
 storageClassName: "csi-ko1-ssd"
 accessMpdes:
```

```
  - ReadWriteOnce
resources:
 requests:
  storage: 30Gi
```

Если мы создадим такой вот PersistentVolumeClaim, то нам выделят из облака диск запрашиваемого объема. Этот диск будет связан с PVC, а он в свою очередь будет примонтирован к Pod'y, контейнеру и приложению.

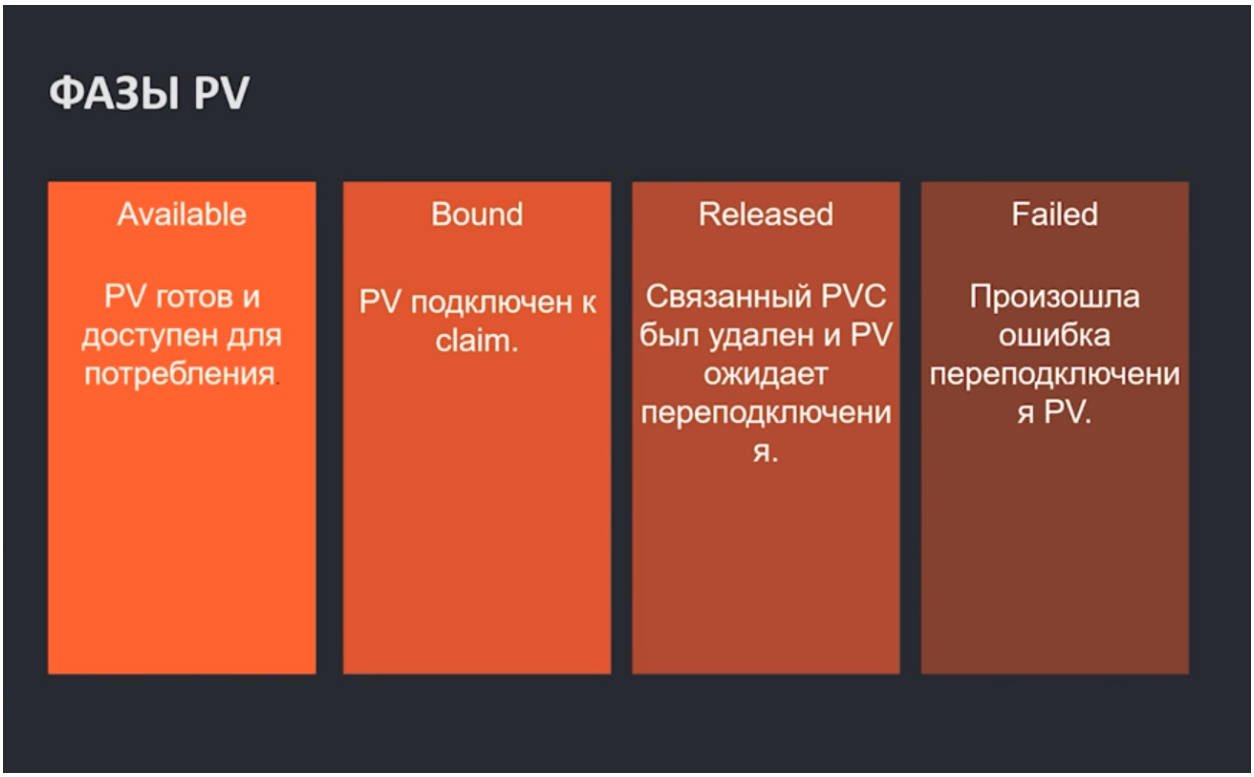
## > Фазы PersistentVolume:

**Available** (доступный)

**Bound** (подключенный)

**Released** (изданный)

**Failed** (поврежденный)

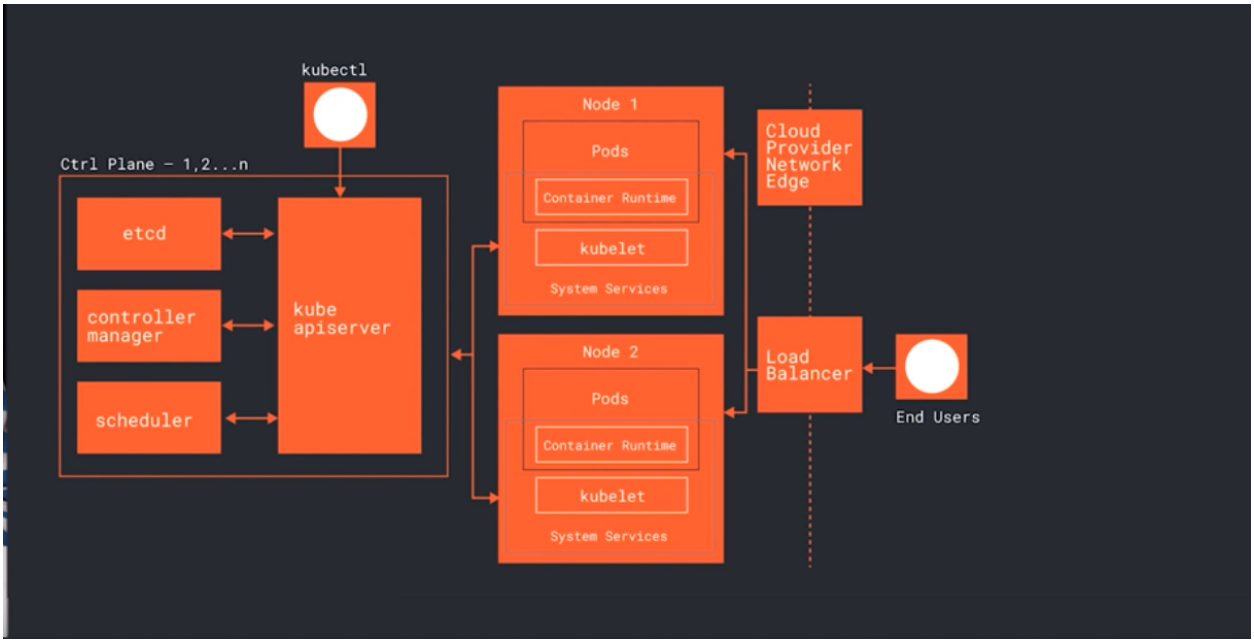


# > Архитектура Kubernetes

Kubernetes - это кластерная система, которая состоит из Ctrl Plane (master Node) и Worker Node, на которых размещается наша нагрузка.

В Ctrl Plane (master Node) важные компоненты, это:

- *etcd*
- *controller manager*
- *scheduler*
- *kube apiserver* - можно считать самым важным, т.к. через него проходят все взаимодействия.
- *kubectl* - утилита командной строки, с помощью которой пользователь взаимодействует с кластером Kuber'a.




## > kube apiserver

- внешний REST API интерфейс;

- все клиенты и компоненты кластера взаимодействуют друг с другом через kube apiserver;

- выступает как единая точка входа в кластер, обрабатывая запросы на аутентификацию, авторизацию, валидацию запросов, изменения и контроль.

## > etcd

- выступает в качестве кластерного хранилища данных;

- обеспечивает надежное консистентное, высокодоступное key-value хранилище, в котором хранится вся конфигурация кластера;

- хранит объекты и информацию о конфигурации;

- использует кворумный алгоритм Raft.

## > controller manager

- реализует логику за контроллерами Kubernetes;

- это он говорит: "Эй, мне нужно поднять еще несколько подов"

## > scheduler

- оценивает требования приложений и пытается их разместить на подходящем сервере;

- оперирует лейблами, требованиями, лимитами по ресурсам, политиками affinity и т.д.

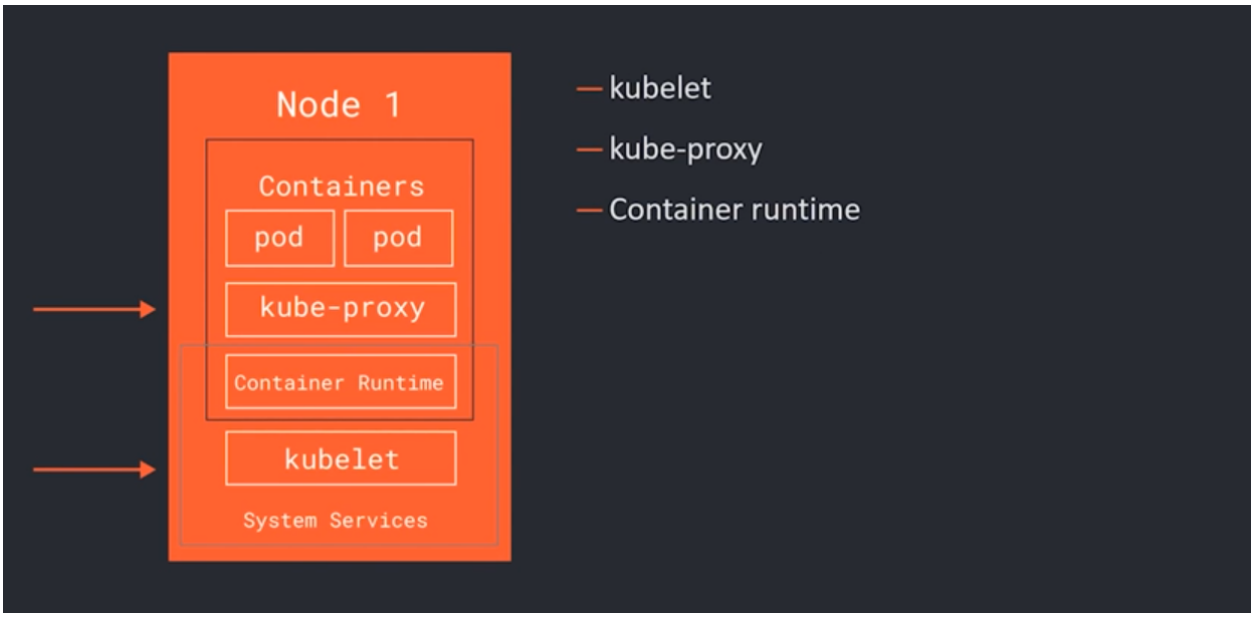
## > Архитектура Nodes

Здесь у нас три важных компонента:

kubelet - агент, работающий на каждом узле и управляющий состоянием Pods. Он опрашивает kube API на предмет подов, которые должен запустить.

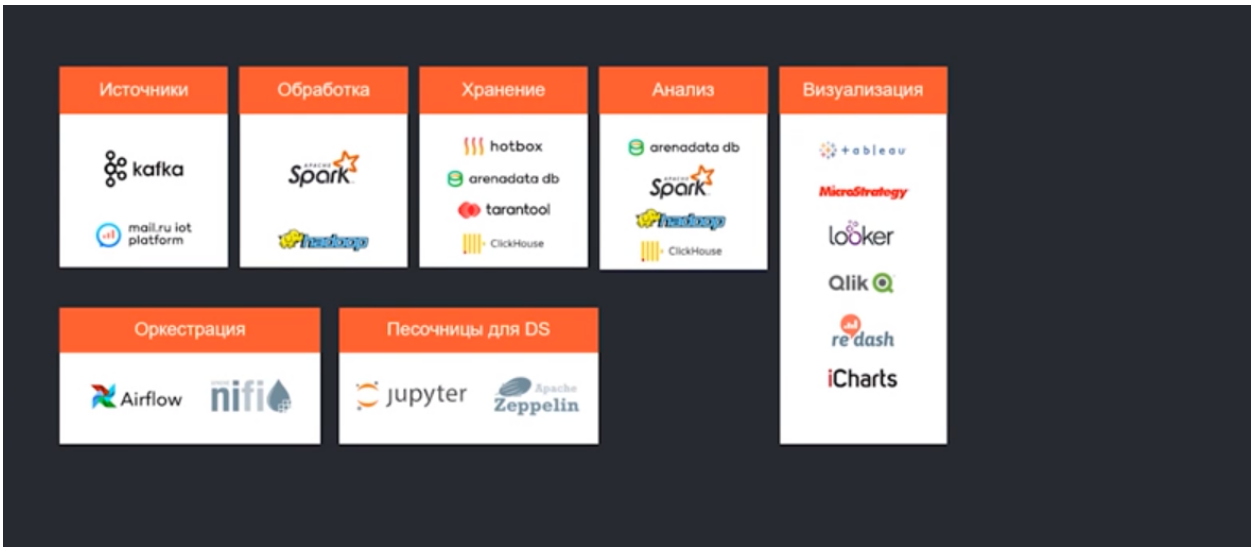
kube-proxy - управляет сетевыми правилами на каждом узле, обеспечивает connection-forwarding и load-balancing для сервисов Kubernetes. В большинстве случаев используется iptables (в современных кластерах можно использовать ipvs)

container runtime - докер runtime, отвечает за запуск приложений, упакованных в контейнер.




## > Big Data + K8S

У нас в облаках есть возможность выстроить комплексную платформу для работы с данными, которая будет полным аналогом традиционного подхода.



Однако, если вы хотите оптимально использовать возможности облаков и не терять в качестве, устойчивости, рекомендуем использовать другую архитектуру. В рамках этого подхода, данные которые мы храним в Hadoop/HDFS, мы храним в S3.

Преимущества хранения данных в S3:

- упрощение процесса хранения и передачи данных с отказоустойчивостью 99,9999;
- экономия средств: S3 дешевле HDFS;
- разделение Storage и Compute-мощностей;
- независимый скейлинг Storage и Compute

## > Spark + K8S

Мы сложили данные в S3, но само S3 не умеет обрабатывать данные. Здесь в дело вступает Spark, который мы запускаем в Kubernetes. Он нам позволяет решать задачу обработки данных.

Spark + K8S это:

- изоляция сред (контейнеризация и depency management);

- гибкое масштабирование;

- Kubernetes Operator for Spark - Kubernetes native way;

- Spark Operator позволяет решить проблемы с доступом к логам, получением текущего статуса и состояния приложения.

Полезные ссылки. Spark + K8S

— Разворачиваем приложение на Apache Spark в Kubernetes. Пошаговый рецепт;

— https://github.com/stockblog/webinar_spark_k8s;

— https://www.lightbend.com/blog/how-to-manage-monitor-spark-on-kubernetesintroduction-spark-submit-kubernetes-operator;

— https://github.com/GoogleCloudPlatform/spark-on-k8soperator/blob/master/docs/quick-start-guide.md

## > Presto + Hive Metastore + K8S

Мы обработали данные, и теперь нам необходимо предоставить SQL-доступ. Для этого часто используется Presto. Связка Presto + Hive Metastore + K8S:

- решает задачу ad-hoc аналитики;

- позволяет получить доступ к данным в различных системах и источниках;

- предоставляет возможность горизонтального масштабирования, масштабирования воркеров на основе HPA;

- Presto K8S Operator от Starbust или Helm chart.

Полезные ссылки. Presto + K8S
— https://blog.starburstdata.com/technical-blog/presto-on-kubernetes;
— https://docs.starburst.io/latest/k8s.html;
— https://github.com/falarica/steerd-presto-operator;
— https://github.com/joshuarobinson/trino-on-k8s;
— https://joshua-robinson.medium.com/presto-powered-s3-data-warehouse-onkubernetes-aea89d2f40e8

## > Superset + K8S

Далее нам необходимо решить задачу с BI и визуализацией. Это Superset + K8S:

- cloud-native;

- большой выбор встроенных инструментов для создания дашбордов;

- его можно использовать как SQL IDE;

- есть Helm chart.

Полезные ссылки. Superset + K8S
— https://dropbox.tech/application/why-we-chose-apache-superset-as-our-data-explorationplatform;
— https://superset.apache.org/docs/installation/installing-superset-from-scratch;

— https://apache-superset.readthedocs.io/en/latest/installation.html;
— https://superset.apache.org/docs/databases/installing-database-drivers;
— https://www.metabase.com/docs/latest/operations-guide/running-metabase-onkubernetes.html

## > Airflow + K8S

- Kubernetes Executor – создаем воркеры по мере необходимости, экономим ресурсы во время простоя;

- Kubernetes Executor – необходимо принимать во внимание время на старт подов;

- KubernetesPodOperator – может работать с CeleryExecutor, дает больше гибкости;

- Храним логи в S3.

Полезные ссылки. Airflow + K8S
— https://towardsdatascience.com/a-journey-to-airflow-on-kubernetes-472df467f556
— https://medium.com/uncanny-recursions/introduction-to-kubernetesexecutor-and-kubernetespodoperatorae9bb809e3b3
— https://airflow.apache.org/docs/apache-airflow/stable/kubernetes.html
— https://airflow.apache.org/docs/apache-airflow/stable/executor/kubernetes.html
— https://airflow.apache.org/docs/apache-airflow-providers-cncf-kubernetes/stable/operators.html#howtooperator-kubernetespodoperator
— https://github.com/apache/airflow/tree/master/chart

## > Amundsen + K8S

- Data Discovery Platform;

- Повышает продуктивность пользователей платформы;

- Помогает демократизировать доступ к данным, share tribal knowledge;

- Интеграция с Airflow;

- Есть Helm chart

Полезные ссылки. Amundsen + K8S
—https://www.amundsen.io/amundsen/
—https://www.amundsen.io/amundsen/k8s_install/

—https://github.com/amundsen-io/amundsen/issues/53
—https://eng.lyft.com/open-sourcing-amundsen-a-data-discovery-and-metadataplatform-2282bb436234
—https://eng.lyft.com/amundsen-1-year-later-7b60bf28602

## > JupyterHub + K8S

- Проведение индивидуальных экспериментов;

- Изолированность окружений, нагрузок, экспериментов;

- Эффективное использование ресурсов, автоматическое выключение простаивающих;

- Подробная инструкция и Helm chart.

Полезные ссылки. JupyterHub + K8S
— https://zero-to-jupyterhub.readthedocs.io/en/latest/jupyterhub/installation.html
— https://zero-to-jupyterhub.readthedocs.io/en/latest/jupyterhub/customization.html

## > Kuberflow или Mlfow

- База данных ML-экспериментов и моделей с кодом, параметрами, результатами;

- Удобный вывод ML-моделей в прод;

- Ускоряет работу DS и DE, повышает эффективность, позволяет получить полную отдачу от результатов DS;

- Kubeflow — Kubernetes native, включает в себя Jupyter Notebooks;

- Mlflow — более стабильный, почти не дружит с Kubernetes.

Полезные ссылки. Kuberflow, Mlflow
— https://medium.com/weareservian/the-cheesy-analogy-of-mlflow-andkubeflow-715a45580fbe
— MLOps без боли в облаке. Разворачиваем Kubeflow – https://www.youtube.com/watch?v=fZ-g2TjhhGE&t
— https://github.com/stockblog/webinar_kubeflow

— https://www.kubeflow.org/docs/started/kubeflow-overview/
— https://mlflow.org/docs/latest/quickstart.html
— https://mlflow.org/docs/latest/projects.html#run-an-mlflow-project-onkubernetes-experimental
— https://github.com/imaginea/mlflow-on-k8s

## > Data + Kubernetes

- S3 – отвечает за хранение;
- Spark – “молотилка” данных;
- Presto + Hive Metastore – ad hoc аналитика, схемы данных;
- Superset – BI, data exploration, SQL IDE;
- Airflow – оркестратор с возможностью масштабирования в k8s;
- Amundsen – data discovery platform;
- JupyterHub – проведение изолированных экспериментов;
- Kubeflow, MLflow – задачи MLOps, трекинг моделей, экспериментов.

## > Tarantool + K8S

Tarantool - это сверхбыстрая база. Она упрощает развертывание и эксплуатацию кластерных приложений.
- Сокращает Time-to-market решений;
- Есть K8S-оператор;
- Решает задачу автоматического добавления узлов, решардинга

Полезные ссылки. Tarantool + K8S
— https://www.tarantool.io/ru/kubernetes/
— https://github.com/tarantool/tarantool-operator/
— https://habr.com/ru/company/mailru/blog/465823/
— https://www.highload.ru/moscow/2019/abstracts/6029