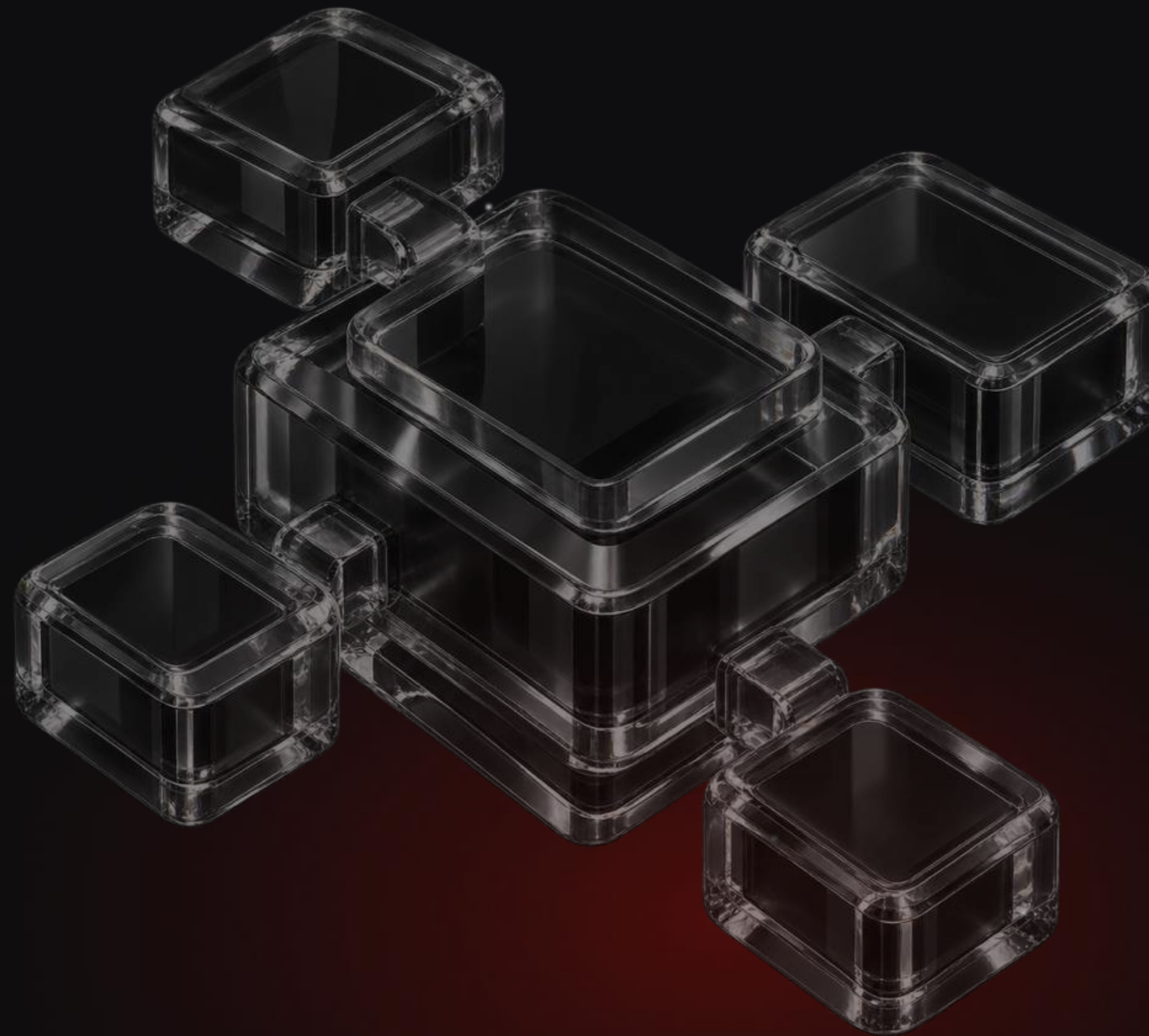


Жизнь после **Managed Spark**

Как мы сделали Spark **управляемым runtime**
на K8s как часть ML-платформы

Салман Даниил

MLOps in CVM Team



Масштабы

В нашем CVM отделе на **30+** человек инженеров:

в проде в среднем

100+

активных DAG на один инстанс AF;

■ ~50% работают ежедневно;

~1000

Spark applications / месяц;

до

2 TB

данных за один запуск;

до старта миграции в k8s:

8

Spark-кластеров,

55

нод,

~1000

vCPU,

~6 TB

RAM.

Сегодня разберём

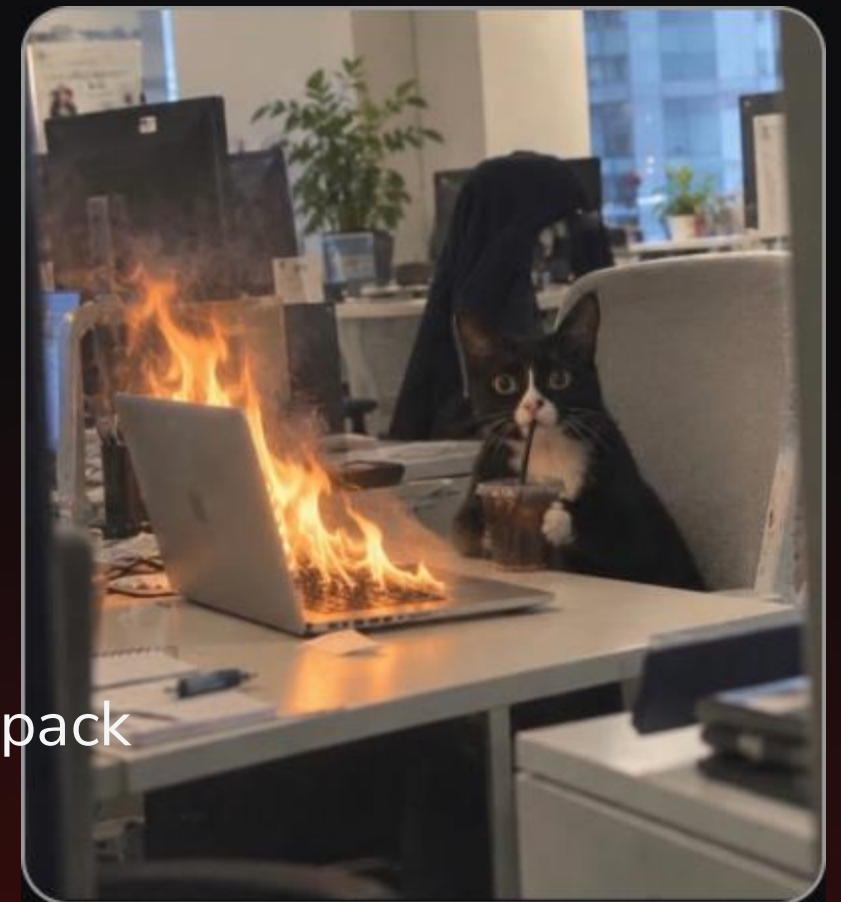
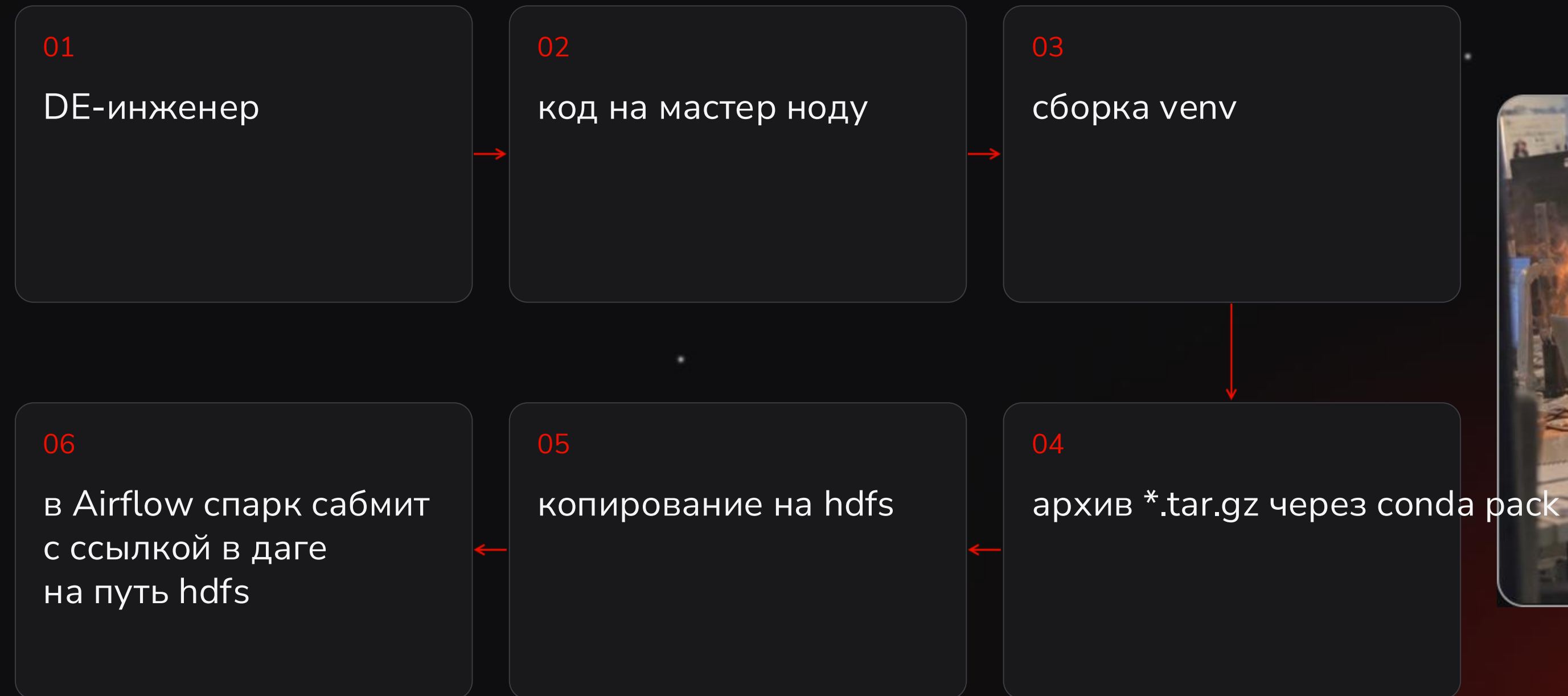
- В чем разница между managed Spark и Spark в k8s
- Почему поставить Spark Operator это только начало
- Какой стек мы использовали, чтобы закрыть наши потребности в Spark
- Как дать “воздух” пользователям в работе и не закапывать их инфраструктурными проблемами

Что вы получите к концу доклада:

- Разберётесь в том какие преимущества дает Spark в k8s и поймете надо ли оно вам
- Узнаете, как контролировать этого “зверя” и настроить правильно процессы на старте
- Небольшой CookBook по тому, как построить мониторинг Spark в k8s
- Унесете интересные подходы и посмотрите на чужой опыт одной из команд крупнейшего ритейлера

Как мы жили в Managed Spark

Pipeline работы DE инженера раньше:



Было принято тактическое решение “мигрировать” и мы его придерживались

Есть два пути:

Использовать SparkSubmitOperator в AF

чтобы просто указывать `--master k8s://` и `image` вместо `--master yarn`

(путь был отвергнут практически сразу, тоже появляются проблемы с миграцией и не получаем преимуществ k8s)

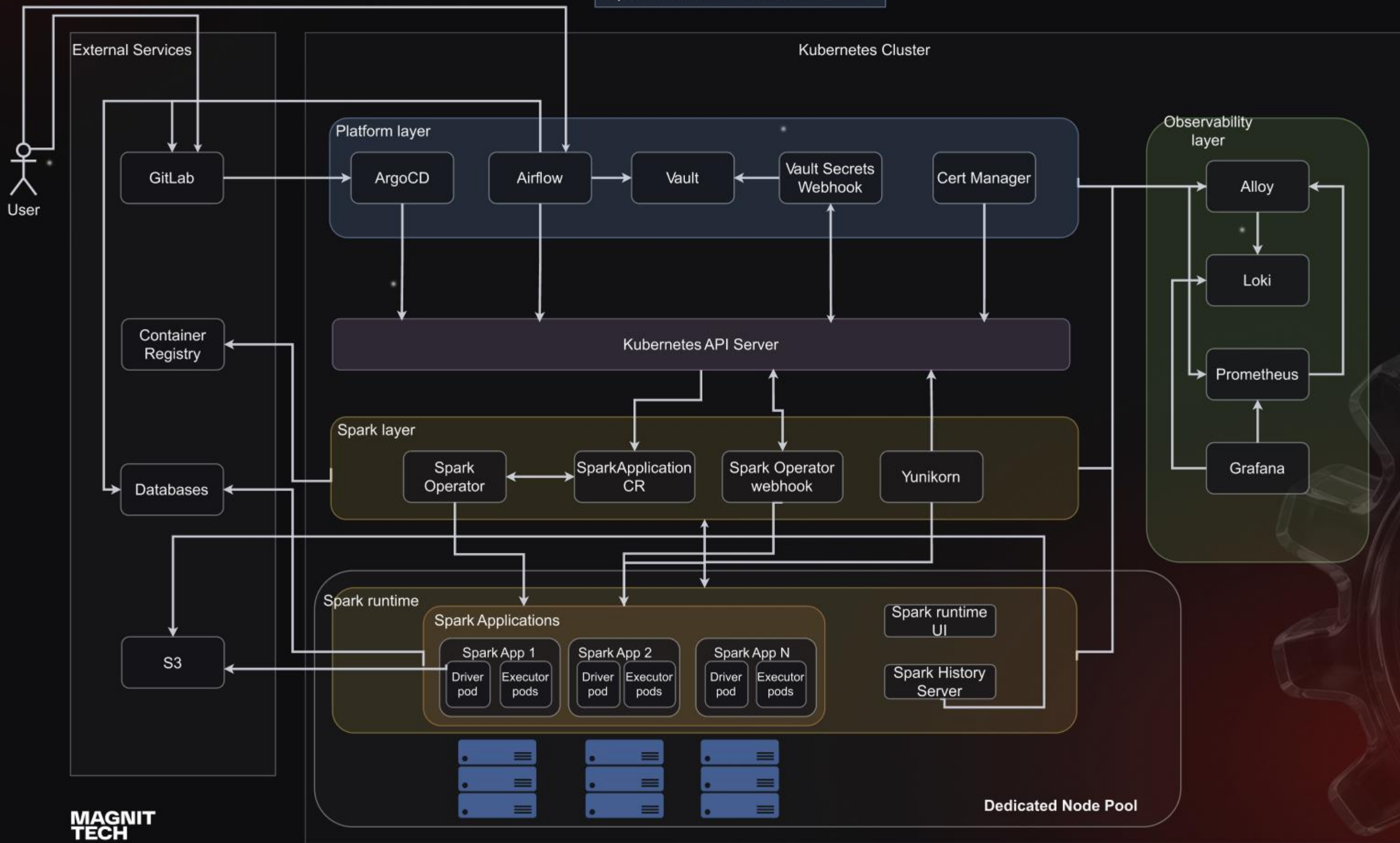
Пишем код → билдим образ → описываем Spark Application через SparkKubernetesOperator и кайфуем (или нет?)

SparkApplication → Spark Operator → Driver + Executors → FINISH ?

Что у нас получилось спустя
время? **Архитектурная схема**



Spark Runtime in k8s architecture



Где заканчивается платформа и начинается пользователь

Платформа:

- Spark runtime;
- base image;
- CI/CD;
- SparkApplication defaults;
- scheduler;
- node placement;
- secrets;
- observability.

Пользователь:

- PySpark-код;
- бизнес-логика;
- проектные зависимости;
- параметры задачи;
- DAG/schedule.

Base spark images

Spark 3.0.3 | Spark 3.3.3 | Spark 3.5.5 | Spark 4.0.0

В base image уже есть

- Spark / Java / Python / Scala
- pinned Python packages
- JDBC / S3 / Iceberg / Kafka JARs
- общие системные зависимости
- платформенные настройки

Docker file DE инженера:

```
FROM registry/.../spark-base:3.0.3-py3.9

COPY requirements.txt .
RUN pip install -r requirements.txt
COPY src/ /opt/spark/app/
```

CI/CD: что остаётся сделать инженеру

git push



CI

- build Spark application image
- push image
- publish DAG
- optionally deploy additional components

gitlab-ci - GitLab CI Configuration file (ci.json)

```
1 stages:
2 |   - feature
3 |   - main
4
5 default:
6 |   timeout: 30m
7
8 variables:
9 |   K8S_ENV: "${DEPLOY_ENV}"
10
11 include:
12 |   - project: "../iac/ci"
13 |     ref: "main"
14 |     file: "/template/jobs/cicd_pipeline.yml"
15 |   - local: "../templates/feature_airflow.yaml"
16 |   - local: "../templates/feature_image_build.yaml"
17 |   - local: "../templates/prod_airflow.yaml"
18 |   - local: "../templates/prod_image_build.yaml"
```

Base spark images

До

```
RUN set -eux; \
# Hadoop S3A (Hadoop 3.4.x uses AWS SDK v2)
curl -fsSL -o ${SPARK_HOME}/jars/hadoop-aws-${HADOOP_AWS_VER}.jar \
https://repo1.maven.org/maven2/org/apache/hadoop/hadoop-aws/${HADOOP_AWS_VER}/hadoop-aws-${HADOOP_AWS_VER}.jar; \
\
VER=$AWS_SDK_V2_BUNDLE_VER && \
URL=$(case "$VER" in 1*) echo "https://repo1.maven.org/maven2/com/amazonaws/aws-java-sdk-bundle/${AWS_SDK_V2_BUNDLE_VER}/aws-java-sdk-bundl
curl -fsSL -o "${SPARK_HOME}/jars/aws-sdk-java-bundle-${AWS_SDK_V2_BUNDLE_VER}.jar" "$URL" && \
\
SPARK_SHORT=$(echo "$SPARK_VERSION" | cut -c1-3) && \
curl -k -L "https://repo1.maven.org/maven2/org/apache/iceberg/iceberg-spark-runtime-${SPARK_SHORT}_${SCALA_BINARY}/${ICEBERG_VER}/iceberg
-o ${SPARK_HOME}/jars/iceberg-spark-runtime-${SPARK_SHORT}_${SCALA_BINARY}-${ICEBERG_VER}.jar; \
\
# JDBC drivers
curl -k -L https://repo1.maven.org/maven2/com/clickhouse/clickhouse-jdbc/0.6.0/clickhouse-jdbc-0.6.0-all.jar \
-o ${SPARK_HOME}/jars/clickhouse-jdbc-0.6.0-all.jar; \
curl -k -L https://repo1.maven.org/maven2/io/trino/trino-jdbc/475/trino-jdbc-475.jar \
-o ${SPARK_HOME}/jars/trino-jdbc-475.jar; \
curl -k -L https://repo1.maven.org/maven2/org/postgresql/postgresql/42.7.1/postgresql-42.7.1.jar \
-o ${SPARK_HOME}/jars/postgresql-42.7.1.jar; \
curl -k -L https://repo1.maven.org/maven2/com/microsoft/sqlserver/mssql-jdbc/12.4.2.jre11/mssql-jdbc-12.4.2.jre11.jar \
-o ${SPARK_HOME}/jars/mssql-jdbc-12.4.2.jre11.jar; \
curl -k -L https://repo1.maven.org/maven2/com/oracle/database/jdbc/ojdbc8/23.3.0.23.09/ojdbc8-23.3.0.23.09.jar \
-o ${SPARK_HOME}/jars/ojdbc8-23.3.0.23.09.jar; \
\
# Spark Kafka integration MUST match Spark/Scala
curl -k -L https://repo1.maven.org/maven2/org/apache/spark/spark-sql-kafka-0-10_${SCALA_BINARY}/${SPARK_VERSION}/spark-sql-kafka-0-10_${S
-o ${SPARK_HOME}/jars/spark-sql-kafka-0-10_${SCALA_BINARY}-${SPARK_VERSION}.jar; \
curl -k -L https://repo1.maven.org/maven2/org/apache/spark/spark-token-provider-kafka-0-10_${SCALA_BINARY}/${SPARK_VERSION}/spark-token-p
-o ${SPARK_HOME}/jars/spark-token-provider-kafka-0-10_${SCALA_BINARY}-${SPARK_VERSION}.jar; \
```

Base spark images

После

```
FROM docker.io/maven:3.9-eclipse-temurin-${JAVA_VERSION} AS maven-runtime-jars

ARG SPARK_VERSION
ARG SCALA_BINARY
ARG HADOOP_VERSION
ARG ICEBERG_VERSION

WORKDIR /build

COPY ./runtime-dependencies/pom.xml ./pom.xml

RUN --mount=type=cache,target=/root/.m2 \
    mvn -B -ntp clean package \
        -Dspark.version="${SPARK_VERSION}" \
        -Dscala.binary.version="${SCALA_BINARY}" \
        -Dhadoop.version="${HADOOP_VERSION}" \
        -Diceberg.version="${ICEBERG_VERSION}"

RUN set -eux; \
    test -d /build/target/runtime-jars; \
    echo "Runtime JARs:"; \
    find /build/target/runtime-jars \
        -maxdepth 1 \
        -type f \
        -name '*.jar' \
        -printf '%f\n' \
        | sort
```

Airflow: до и после платформенной абстракции

Стандартный ДАГ

```
1 run = SparkKubernetesOperator(
2     dag=dag,
3     task_id="run_spill_generator_run",
4     namespace="spark-dev",
5     kubernetes_conn_id="kubernetes_default",
6     get_logs=True,
7     delete_on_termination=True,
8     reattach_on_restart=True,
9     random_name_suffix=True,
10    template_spec={
11        "apiVersion": "sparkoperator.k8s.io/v1beta2",
12        "kind": "SparkApplication",
13        "spec": {
14            "type": "Python",
15            "mode": "cluster",
16            "sparkVersion": "4.0.1",
17            "image": IMAGE,
18            "imagePullPolicy": "Always",
19            "deps": {
20                "jars": [
21                    "local:///opt/spark/jars/aws-sdk-java-bundle-2.24.6.jar",
22                    "local:///opt/spark/jars/hadoop-aws-3.4.1.jar",
23                ],
24                "packages": [
25                    f"org.apache.iceberg:iceberg-spark-runtime-4.0_2.13:{ICEBERG_VERSION}",
26                    f"org.apache.iceberg:iceberg-aws-bundle:{ICEBERG_VERSION}",
27                ],
28            },
29        },
30    }
```

Airflow: до и после платформенной абстракции

Стандартный ДАГ

```
29     "sparkConf": {
30         "spark.sql.shuffle.partitions": "{{ dag_run.conf.get('shuffle_partitions', '20') }}",
31         f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.aws.credentials.provider": "org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider",
32         f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.endpoint": vars.S3_ENDPOINT.get().removeprefix("https://"),
33         f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.endpoint.region": vars.S3_REGION.get(),
34         f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.access.key": vars.S3_ACCESS_KEY.get(),
35         f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.secret.key": vars.S3_SECRET_KEY.get(),
36         "spark.hadoop.fs.s3a.bucket.bucket-name.aws.credentials.provider": "org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider",
37         "spark.hadoop.fs.s3a.bucket.bucket-name.endpoint": "endpoint",
38         "spark.hadoop.fs.s3a.bucket.bucket-name.endpoint.region": vars.S3_PROD_REGION.get(),
39         "spark.hadoop.fs.s3a.bucket.bucket-name.access.key": vars.S3_PROD_ACCESS_KEY.get(),
40         "spark.hadoop.fs.s3a.bucket.bucket-name.secret.key": vars.S3_PROD_SECRET_KEY.get(),
41     },
42     "mainApplicationFile": "local:///opt/spark/work-dir/main.py",
43     "arguments": [
44         "--fv_name_1", "{{ dag_run.conf.get('fv_name_1', 'cheques_visits_v1') }}",
45         "--fv_name_2", "{{ dag_run.conf.get('fv_name_2', 'cheques_art_v1') }}",
46         "--event_timestamp", "{{ dag_run.conf.get('event_timestamp', '2026-04-27') }}",
47         "--mode", "spark-nonreplicated-ssd",
48     ],
49     "driver": {
50         "cores": 1,
51         "coreLimit": "1",
52         "memory": "2g",
53         "serviceAccount": "spark-driver",
54         "nodeSelector": node_selector,
55         "tolerations": [{"key": "dedicated-to-services", "operator": "Equal", "value": NODE_POOL, "effect": "NoSchedule"}],
56     },
```

Airflow: до и после платформенной абстракции

Стандартный ДАГ

```
57 |         |         | "executor": {
58 |         |         |     "instances": 3,
59 |         |         |     "cores": 2,
60 |         |         |     "memory": "2g",
61 |         |         |     "serviceAccount": "spark-driver",
62 |         |         |     "nodeSelector": node_selector,
63 |         |         |     "tolerations": [{"key": "dedicated-to-services", "operator": "Equal", "value": NODE_POOL, "effect": "NoSchedule"}],
64 |         |         | },
65 |         |     },
66 |     },
67 | )
```

Airflow: до и после платформенной абстракции

После упрощения

```
1 r = SparkKubernetesOperator(
2     dag=dag,
3     task_id="run",
4     template_spec={
5         "spec": {
6             "sparkVersion": "4.0.1",
7             "image": IMAGE,
8             "sparkConf": {
9                 f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.endpoint": vars.S3_ENDPOINT.get().removeprefix(
10                     "https://"
11                 ),
12                 f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.endpoint.region": vars.S3_REGION.get(),
13                 f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.aws.credentials.provider": "com.magnit.CustomEnvCredentialsProviderV2",
14                 f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.access.env": "S3_GENERIC_ACCESS_KEY",
15                 f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.secret.env": "S3_GENERIC_SECRET_KEY",
16             },
17             "mainApplicationFile": "local:///opt/spark/work-dir/main.py",
18             "arguments": [
19                 "{{ dag_run.conf['parquet_path'] }}",
20             ],
21             "driver": {
22                 "cores": 1,
23                 "coreLimit": "1",
24                 "memory": "4g",
25                 "env": envs,
26             },
27             "executor": {
28                 "instances": 2,
29                 "cores": 1,
30                 "coreLimit": "2",
31                 "memory": "3g",
32                 "env": envs,
33             },
34         },
35     },
36 )
```

Интеграция AF + Spark и Vault: Секреты и доступы

Что сделали:

- в Airflow добавлен connection “vault_default” к Vault

в Wrapper для Spark мы дописали:

- логику по работе с Secrets Webhook (аннотации заезжают во все поды)
- typeChecking, контекстные подсказки
- зашили default Spark conf для инфраструктурных параметров

Что получили:

- Vault - единая точка правды для секретов
- Секреты приезжают в Airflow, Spark driver и executors
- Не кладём credentials в DAG, SparkConf, не зашиваем в образы и код, не попадают в логи
- Spark ходит в несколько внешних систем (базы данных, несколько бакетов) без лишних трудозатрат
- **Пользователям не надо следить за тем, чтобы “инфраструктурный” код был верно перенесен из шаблона-проекта**

Интеграция AF + Spark и Vault: Секреты и доступы

```
8
7
6 r: SparkKubernetesOperator = SparkKubernetesOperator(
5     dag=dag,
4     task_id="run",
3     template_spec={
2         "spec": {
1             "sparkVersion": "4.0.1",
61             "imAge": IMAGE,
1             "deps": {
2                 "jars": [
3                     "local:///opt/spark/jars/hadoop-aws-3.4.1.jar",
4                     "local:///opt/spark/jars/hadoop-custom-env-credentials-provider-1.0.0-SNAPSHOT.jar",
5                 ]
6             },
7             "sparkConf": {
8                 f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.endpoint": vars.S3_ENDPOINT.get().removeprefix(
9                     "https://"
10                ),
11                 f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.endpoint.region": vars.S3_REGION.get(),
12                 f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.aws.credentials.provider": "com.magnit.CustomEnvCredentialsProvi
13                 f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.access.env": "S3_GENERIC_ACCESS_KEY",
14                 f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.secret.env": "S3_GENERIC_SECRET_KEY",
15             },
16             "mainApplicationFile": "local:///opt/spark/work-dir/main.py",
17             "arguments": [
18                 "{{ dag_run.conf['parquet_path'] }}",
19             ],
20             "driver": {
```

```
~/coding/work/hadoop-custom-env-credentials-provider-example/airflow/dags/ 1
hadoop_custom_env_credentials_provider_example.py 1
```

Argument of type "dict[str, dict[str, str | Dependencies | dict[str, str] | list[str] | DriverSpec | ExecutorSpec]]" cannot be assigned to "imAge" is an undefined item in type "SparkApplicationSpec" basedpyright (reportArgumentType) [61, 13]

Интеграция AF + Spark и Vault: Секреты и доступы

```
8
7
6 r: SparkKubernetesOperator = SparkKubernetesOperator(
5     dag=dag,
4     task_id="run",
3     template_spec={
2         "spec": {
1             "sparkVersion": "4.0.1",
61         "image": IMAGE,
1         "imagePullPolicy"
2         "imagePullSecrets"
3         "imageToLiveSeconds" ark/jars/hadoop-aws-3.4.1.jar",
4         ark/jars/hadoop-custom-env-credentials-provider-1.0.0-SNAPSHOT.jar",
5         import~
6         ifmain~
7         ifmainMS~
8         datetime~
9         time~
10        fromim~
11        f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.endpoint.region": vars.S3_REGION.get(),
12        f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.aws.credentials.provider": "com.magnit.CustomEnvCredentialsProvi
13        f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.access.env": "S3_GENERIC_ACCESS_KEY",
14        f"spark.hadoop.fs.s3a.bucket.{vars.S3_BUCKET_NAME.get()}.secret.env": "S3_GENERIC_SECRET_KEY",
15        },
16        "mainApplicationFile": "local:///opt/spark/work-dir/main.py",
17        "arguments": [
18            "{{ dag_run.conf['parquet_path'] }}",
19        ],
20        "driver": {
```

```
~/coding/work/hadoop-custom-env-credentials-provider-example/airflow/dags/ 1
hadoop_custom_env_credentials_provider_example.py 1
```

Argument of type "dict[str, dict[str, str | Dependencies | dict[str, str] | list[str] | DriverSpec | ExecutorSpec]]" cannot be assigned to "im" is an undefined item in type "SparkApplicationSpec" basedpyright (reportArgumentType) [61, 13]

Что пришлось доработать из OpenSource

Custom S3 Credentials Provider (AWS CredentialsProvider в связке с s3a)

- разные credentials для bucket'ов через S3A per-bucket configuration (как и у SimpleCredProvider);
- Hadoop Configuration хранит имена env variables, а не значения credentials;
- provider читает credentials из окружения driver/executor и передаёт их S3A через AWS SDK v1 или v2 в зависимости от spark runtime;
- Кастомный provider даёт дополнительную security-изоляцию секрета от Hadoop Configuration;
- работает с более старыми версиями hadoop;

Simple AWS Provider: bucket → Hadoop config → credential

Custom S3 Provider: bucket → Hadoop config → **ENV NAME** → **ENV** → credential

Что пришлось доработать из OpenSource

Custom S3 Credentials Provider

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("MyApp") \
    \
    .config("spark.hadoop.fs.s3a.bucket.data.access.env", "AWS_KEY_DATA") \
    .config("spark.hadoop.fs.s3a.bucket.data.secret.env", "AWS_SECRET_DATA") \
    .config("spark.hadoop.fs.s3a.bucket.data.aws.credentials.provider",
            "com.magnit.CustomEnvCredentialsProviderV2") \
    \
    .config("spark.hadoop.fs.s3a.bucket.archive.access.env", "AWS_KEY_ARCHIVE") \
    .config("spark.hadoop.fs.s3a.bucket.archive.secret.env", "AWS_SECRET_ARCHIVE") \
    .config("spark.hadoop.fs.s3a.bucket.archive.aws.credentials.provider",
            "com.magnit.CustomEnvCredentialsProviderV2") \
    \
    .getOrCreate()

# Read from different buckets
df_data = spark.read.parquet("s3a://data/data/")
df_archive = spark.read.parquet("s3a://archive/data/")
```

Что пришлось доработать из OpenSource

VaultHook

(мы назвали это VaultClient)



-
- получение секрета по path/key;
 - caching;
 - masking ***;

Что пришлось доработать из OpenSource

VaultHook (мы назвали это VaultClient)

Usage

```
from apache_airflow_providers_hashicorp_custom import VaultClient

client = VaultClient()

# Create a secret reference - no fetch yet
password = client.secret("path/to/db", "password")
username = client.secret("path/to/db", "username")

# All secrets from "path/to/db" are cached in client after first access
db.connect(username=username.get("default_user"), password=password.get())

# All secrets are cached but masked
print(client.cache["path/to/db"]) # {'password': '***', 'username': '***'}

# Vault fetch occurs only during task execution, not at parse time.
run_pod = KubernetesPodOperator(
    ...,
    env_vars=[
        {"name": "DB_USERNAME", "value": username.get("default-user")},
        {"name": "DB_PASSWORD", "value": password.get()},
    ],
)
```

Рост пользователей и Spark-приложений

- Миграция и self-service увеличили количество запусков
- Появляются задачи разной критичности
- Автоскейлинг работает, но ресурсы могут фрагментироваться

Нода может быть занята на

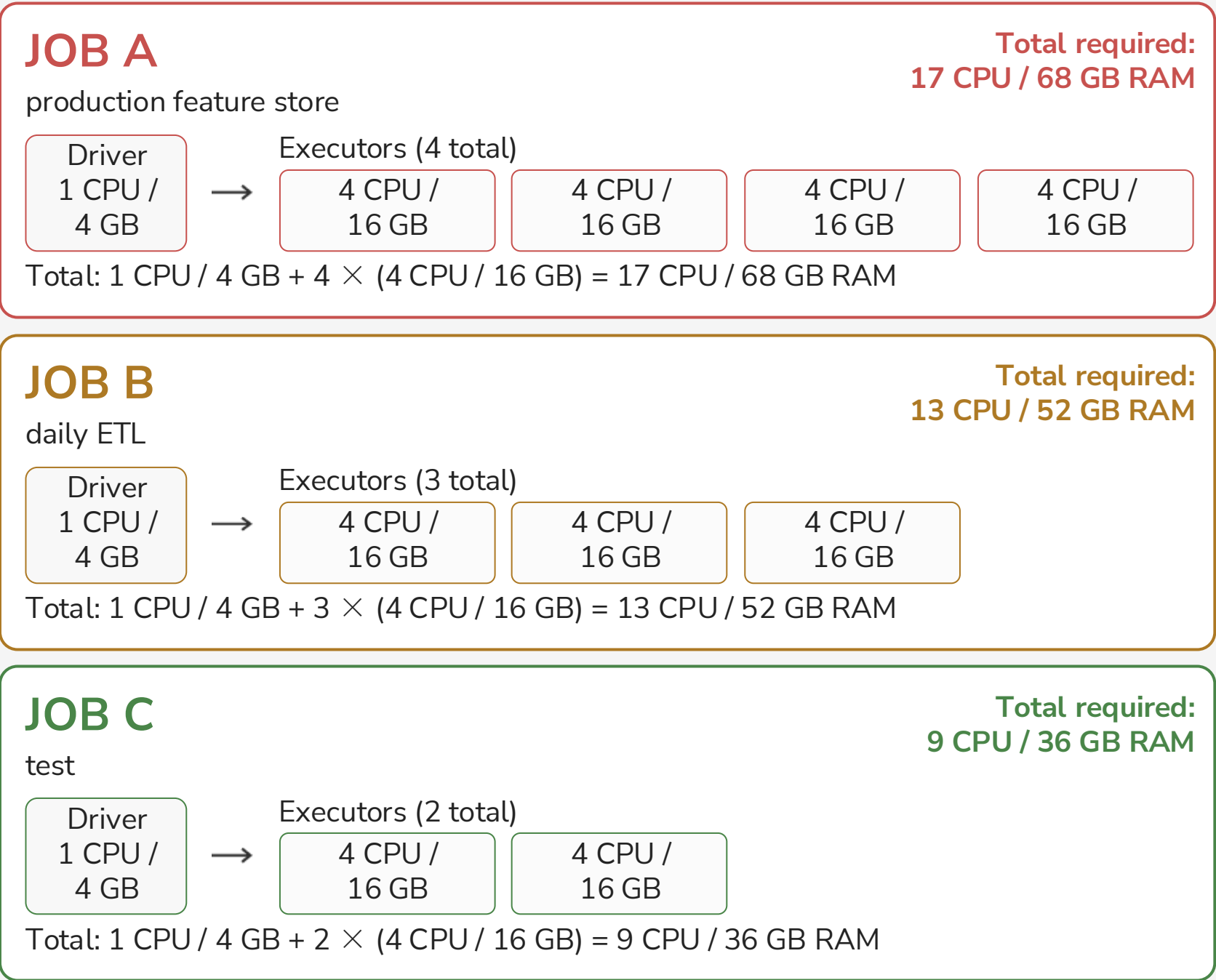
40–50%,

а крупный executor всё равно не помещается

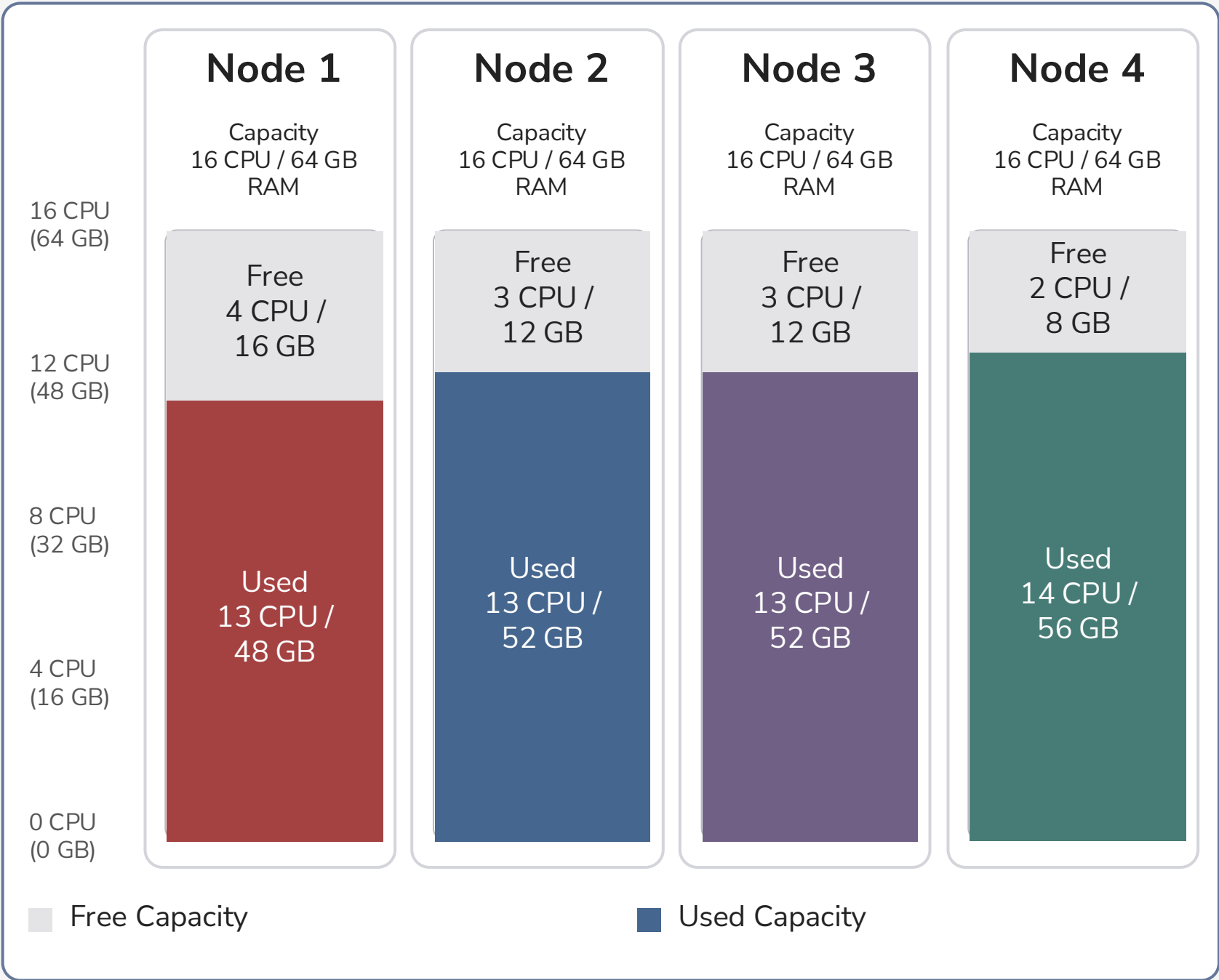
-
- Default k8s scheduler видит прежде всего отдельные pod'ы

Рост пользователей и Spark-приложений

Spark Applications (Queued)

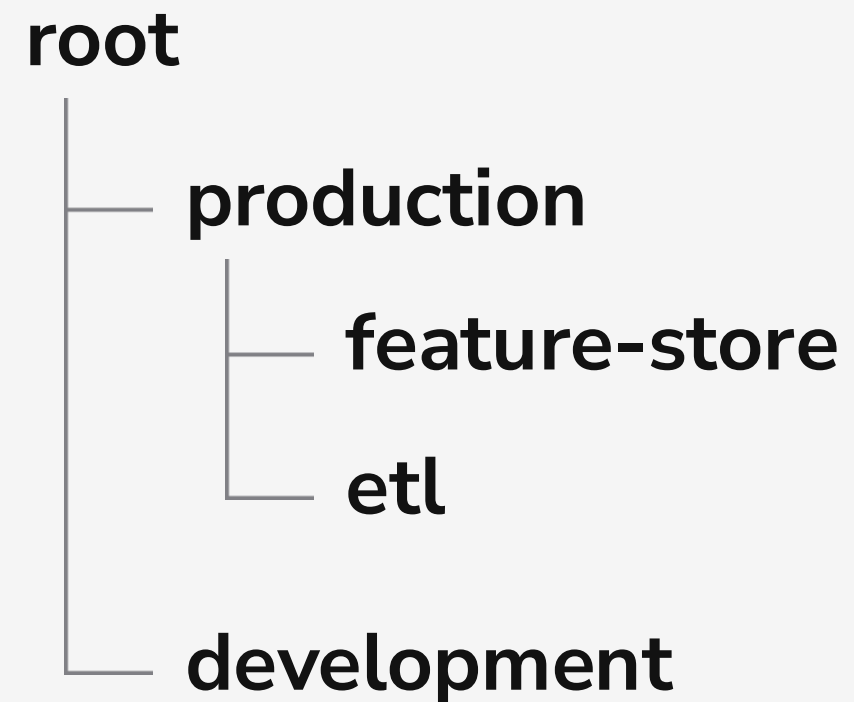


Dedicated Spark Node Pool



Почему YuniKorn

Queues



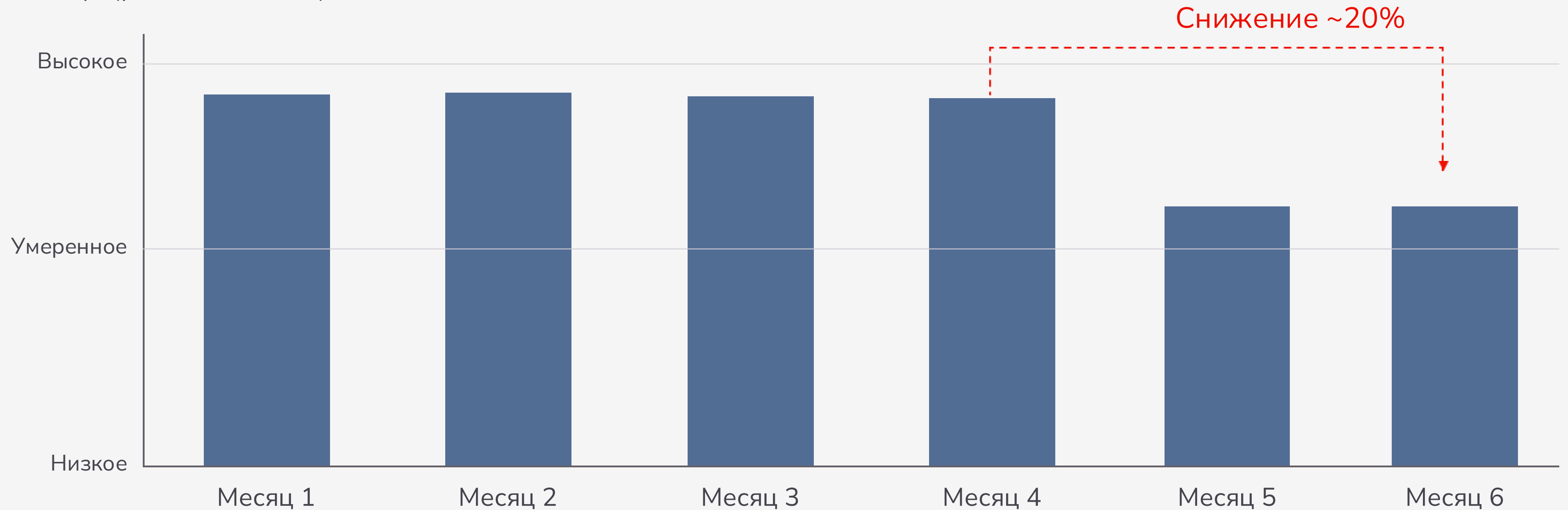
- Quotas

- Gang Scheduling

- Binpacking

Цифры по утилизации после внедрения Yunikorn

Потребление ресурсов
кластера (условные единицы)



Observability

- После runtime/security/scheduling следующий слой – эксплуатация
- мониторинг должен решать проблему “не запускается spark приложение”
- Нужна сквозная наблюдаемость от Airflow до Spark

Observability должна отвечать
«Где именно сломалось?»



Что мониторим как платформенная команда

На уровне Airflow: вообще пытались ли мы запустить задачу?

На уровне Operator: обработал ли контроллер SparkApplication?

На уровне CR: в каком lifecycle статусе находится приложение?

Какие шаги мы сделали на пути к полноценному observability?

YuniKorn

- проверили реально доступные Prometheus metrics
- отказались от несуществующих метрик вроде yunikorn_queue_app
- нормализовали scheduler latency через recording rules
- добавили queue/resource/scheduling rules и alerts

Spark Operator

- использовали встроенные controller metrics
- добавили recording rules для reconcile, workqueue, API errors, saturation, process health
- добавили alerts только на control-plane проблемы, а не на обычный application failure

Какие шаги мы сделали на пути к полноценному observability?

SparkApplication

- включили custom resource metrics через kube-state-metrics.custom-resource-state
- получили lifecycle CR: SUBMITTED / RUNNING / COMPLETED / FAILED / UNKNOWN
- добавили alerts на SubmissionFailed и UnknownState

Spark node pool

- собрали allocatable, requests, actual usage и headroom
- добавили CPU/RAM usage по каждой ноде
- pending/unschedulable workloads
- health dedicated Spark pool

Какие шаги мы сделали на пути к полноценному observability?

Spark runtime

стандартизировали native Prometheus monitoring во всех SparkApplication:

```

```
spark.ui.prometheus.enabled: "true"
spark.metrics.namespace: "spark"
spark.metrics.conf.*.sink.prometheusServlet.class: org.apache.spark.metrics.sink.PrometheusServlet
spark.metrics.conf.*.sink.prometheusServlet.path: /metrics/prometheus
spark.metrics.conf.*.source.jvm.class: org.apache.spark.metrics.source.JvmSource
spark.metrics.appStatusSource.enabled: "true"
```

```

scrape driver :4040:

/metrics/prometheus/
/metrics/executors/prometheus/

проверили это на Spark 3.0.3 / 3.3.3 / 3.5.5 / 4.0.0

Какие шаги мы сделали на пути к полноценному observability?

Logs (loki + alloy)

- доработали Alloy relabeling;
- низкокардинальные признаки оставили labels;
- application/executor/submission данные вынесли в structured metadata;
- разделили driver/executor;
- поправили фильтрацию, чтобы Spark Operator не попадал в runtime logs;
- учли warn и warning.

History Server

добавили Prometheus JMX Exporter как javaagent:

```
—  
-javaagent:/opt/jmx-exporter/jmx_prometheus_javaagent.jar=9404:/opt/jmx-exporter/config.yaml-
```

и много чего еще, что я уже успел забыть...

Какие шаги мы сделали на пути к полноценному observability?

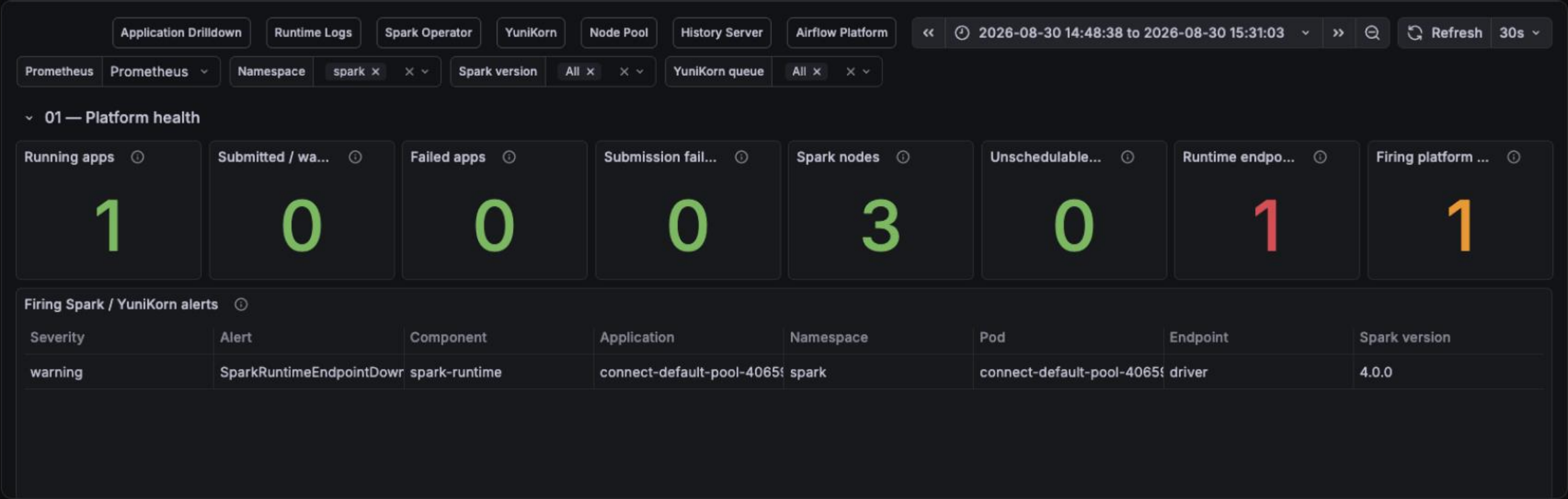
Какие dashboards (10) получились:

Spark Platform Overview

← точка входа при диагностике

- Spark Application Drilldown
- Spark Runtime
- Spark Logs Drilldown
- Spark Operator Control Plane Drilldown
- YuniKorn Scheduling & Queue Drilldown
- Spark Node Pool & Capacity Drilldown
- Spark History Server Drilldown
- Airflow Platform Overview
- Airflow DAG Drilldown

Spark Platform Overview



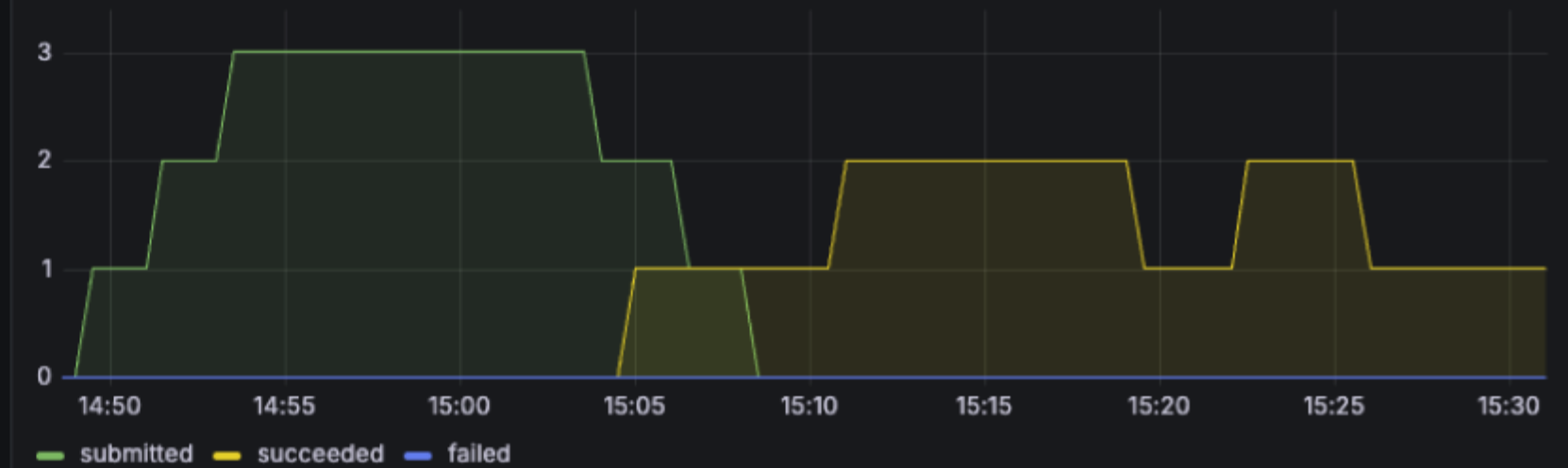
Spark Platform Overview

02 — SparkApplication lifecycle

Applications by state



Operator application activity — 15m



Application start latency



Submission wait — top applications



Spark Platform Overview

03 — Dedicated Spark node pool

CPU — requests vs actual usage



Memory — requests vs working set



Resource headroom



Memory headroom



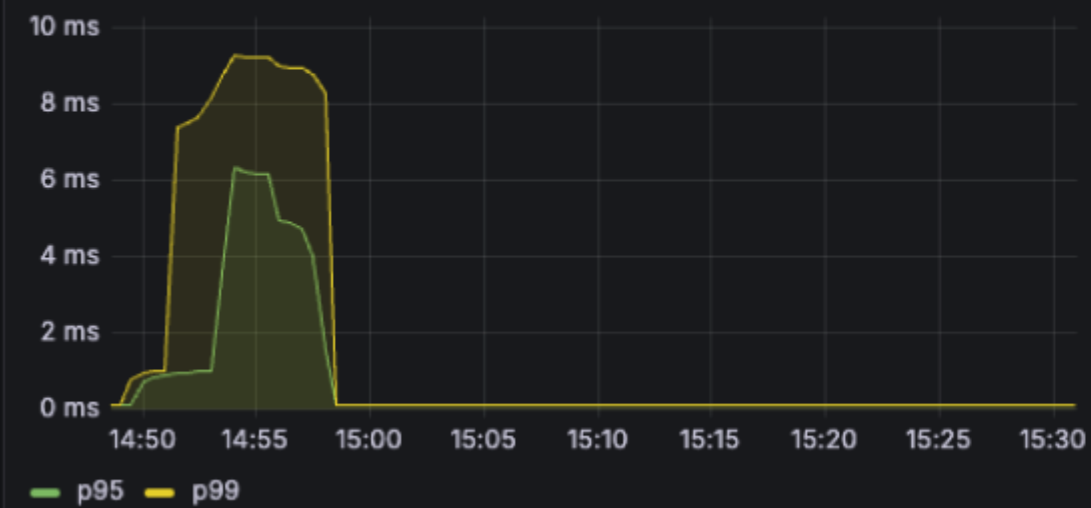
Pending / unschedulable Spark pods



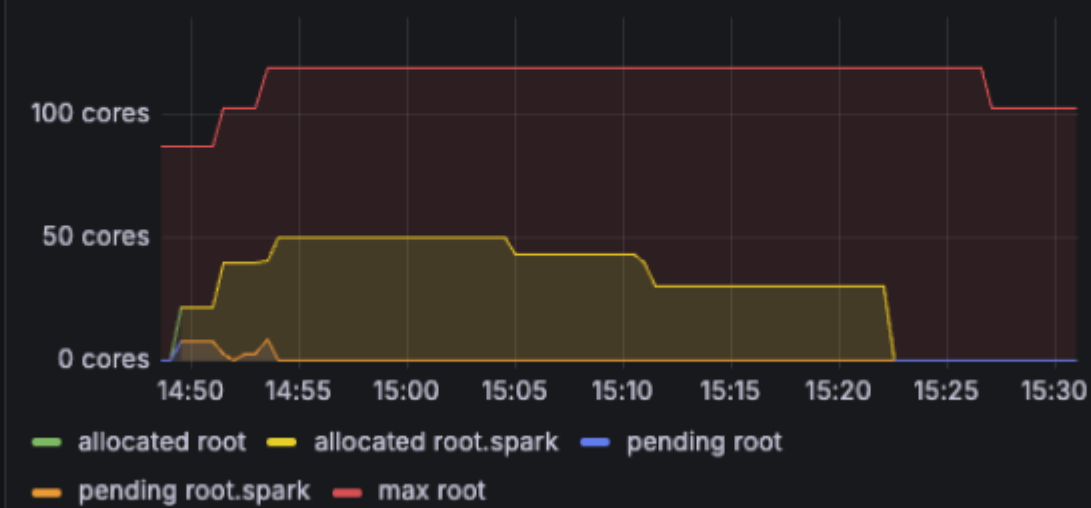
Spark Platform Overview

04 — YuniKorn scheduling

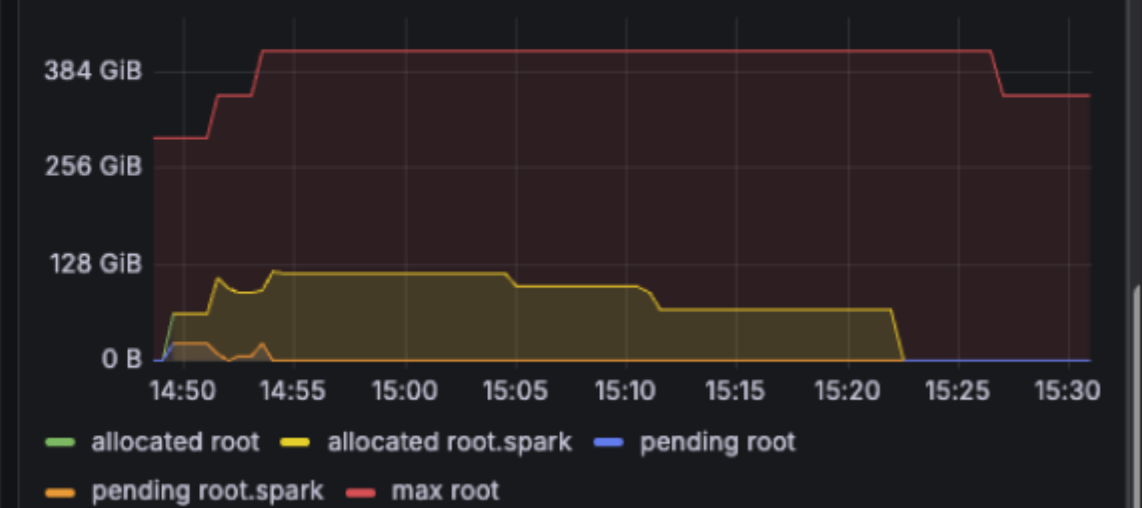
Scheduling latency ⓘ



Queue CPU — allocated / pending / max ⓘ



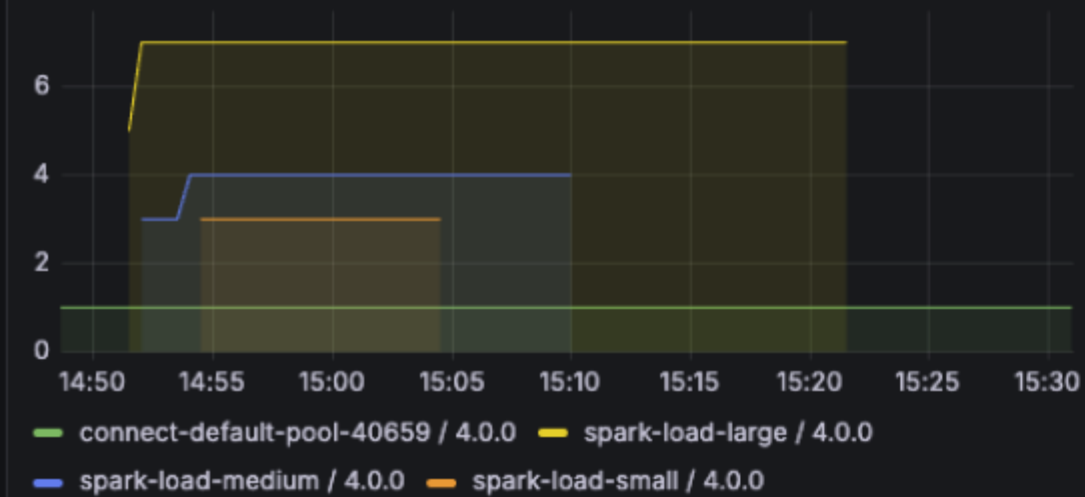
Queue memory — allocated / pending / max ⓘ



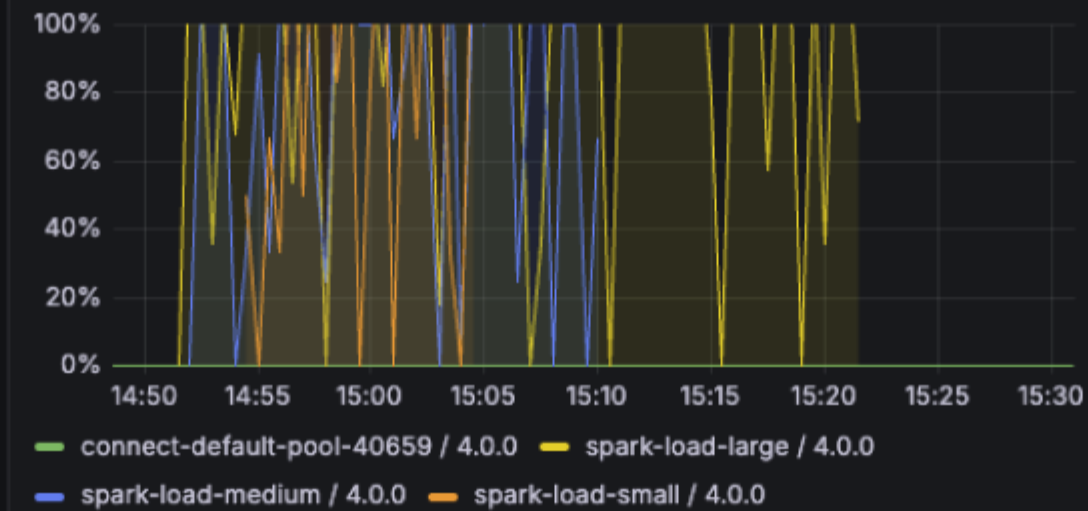
Spark Platform Overview

05 — Spark runtime

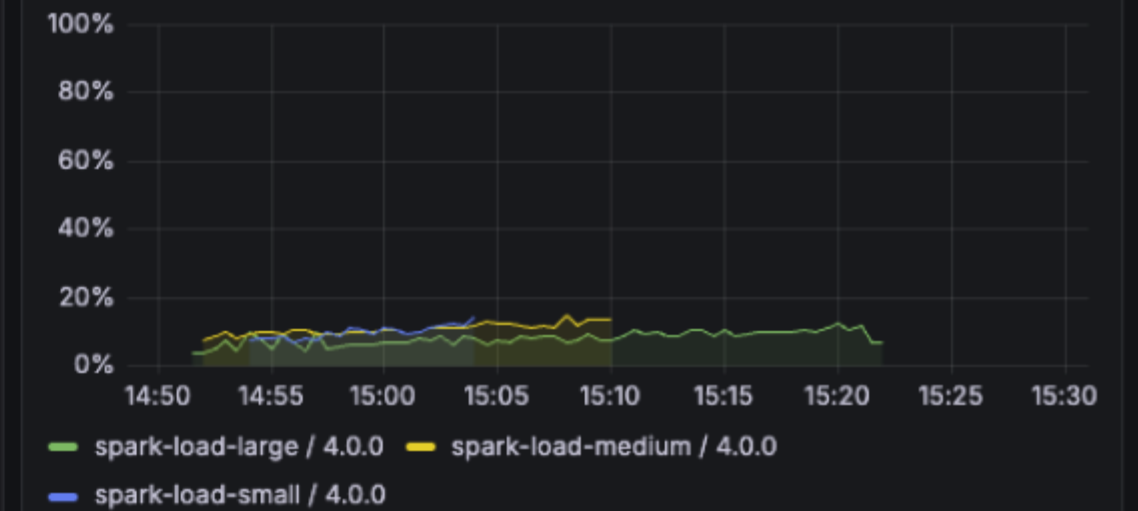
Executors — top applications



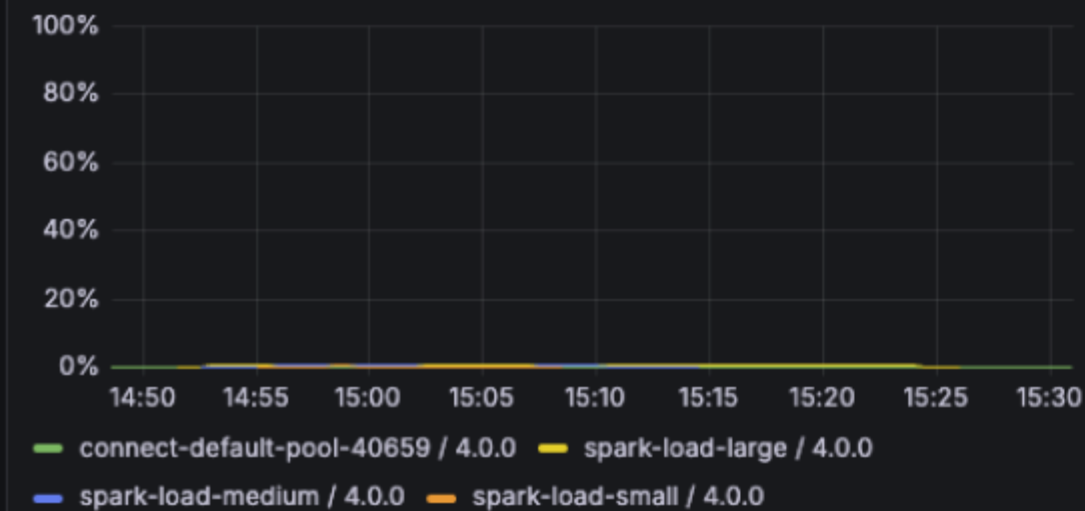
Executor task-slot utilization — top applications



Driver heap usage — top applications



Executor GC wall-time ratio — top applications ⓘ



Input / shuffle throughput

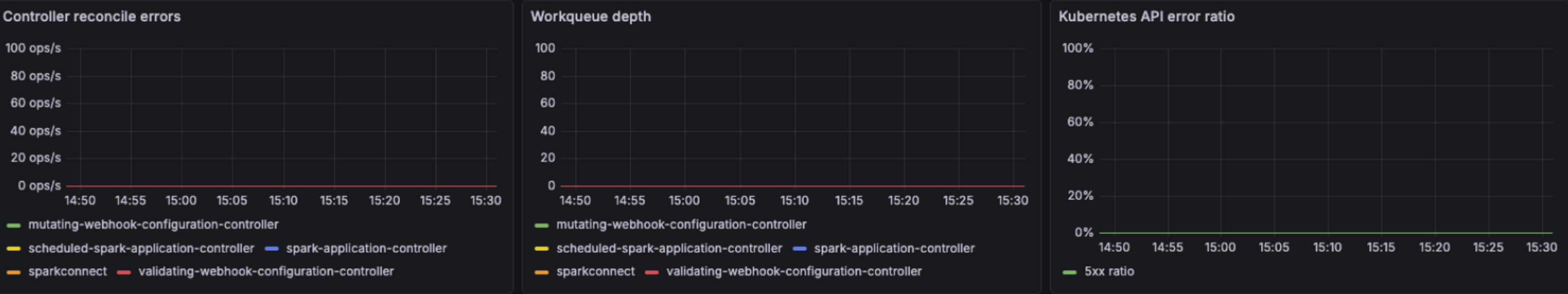


Failed tasks — increase over 10m ⓘ



Spark Platform Overview

06 — Spark Operator control plane



Какие шаги мы сделали на пути к полноценному observability?



Spark Runtime UI vs Spark History Server

Spark UI Index

Active SparkApplications. Page refreshes every 15 seconds.

Namespace	Application	State	Runtime UI
spark	connect-default-pool-40659	RUNNING	open Spark UI
spark	spark-load-large	RUNNING	open Spark UI
spark	spark-load-medium	RUNNING	open Spark UI
spark	spark-load-small	SUBMITTED	open Spark UI

Spark Jobs (?)

User: spark
Started At: 2026/08/30 10:55:55
Total Uptime: 9.7 min
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 783, only showing 781

▶ Event Timeline

▼ Active Jobs (1)

Page: 1

1 Pages. Jump to 1. Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
781	collect at /opt/spark/work-dir/main.py:77 collect at /opt/spark/work-dir/main.py:77 (kill)	2026/08/30 11:05:34	1.0 s	0/2	81/224 (29 running)

Page: 1

1 Pages. Jump to 1. Show 100 items in a page. Go

▼ Completed Jobs (783, only showing 781)

Page: 1 2 3 4 5 6 7 8 >

8 Pages. Jump to 1. Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
780	collect at /opt/spark/work-dir/main.py:77 collect at /opt/spark/work-dir/main.py:77	2026/08/30 11:05:33	0.7 s	1/1	112/112
779	collect at /opt/spark/work-dir/main.py:77 collect at /opt/spark/work-dir/main.py:77	2026/08/30 11:05:33	15 ms	1/1 (2 skipped)	1/1 (224 skipped)
778	collect at /opt/spark/work-dir/main.py:77 collect at /opt/spark/work-dir/main.py:77	2026/08/30 11:05:32	1.0 s	1/1 (1 skipped)	112/112 (112 skipped)
777	collect at /opt/spark/work-dir/main.py:77 collect at /opt/spark/work-dir/main.py:77	2026/08/30 11:05:32	0.7 s	1/1	112/112
776	collect at /opt/spark/work-dir/main.py:77 collect at /opt/spark/work-dir/main.py:77	2026/08/30 11:05:32	19 ms	1/1 (2 skipped)	1/1 (224 skipped)
775	collect at /opt/spark/work-dir/main.py:77 collect at /opt/spark/work-dir/main.py:77	2026/08/30 11:05:31	1.0 s	1/1 (1 skipped)	112/112 (112 skipped)

Выводы

Spark Operator $\neq$ Spark platform;

- После Managed Spark **ответственность** переезжает к вам;
- **Стандартизировать** нужно runtime, delivery, secrets, scheduling, observability;
- Главный продукт платформы – **одинаковый путь** для пользователя, и самое важное – он должен сам желать по нему “ходить” каждый день;
- Kubernetes даёт гибкость, но за неё платим **платформенной сложностью**.

На выходе мы получили:

экономия на ресурсах облака
где-то в среднем

на 15-20%

только на автоскейлинге
и ухода с датапроков в k8s

время выкатки с нуля sparkApp

с 60 до 15

минут (среднее из слов пользователей)

- радость пользователей от отсутствия инфраструктурной нагрузки и лишних действий (не измерима в метриках)
- мониторинг, который экономит множество нервов кликов и часов хождений по логам в поисках проблемы и компонента, который её потенциально вызвал

Спасибо!



Салман Даниил

MLOps in CVM Team

**MAGNIT
TECH**

