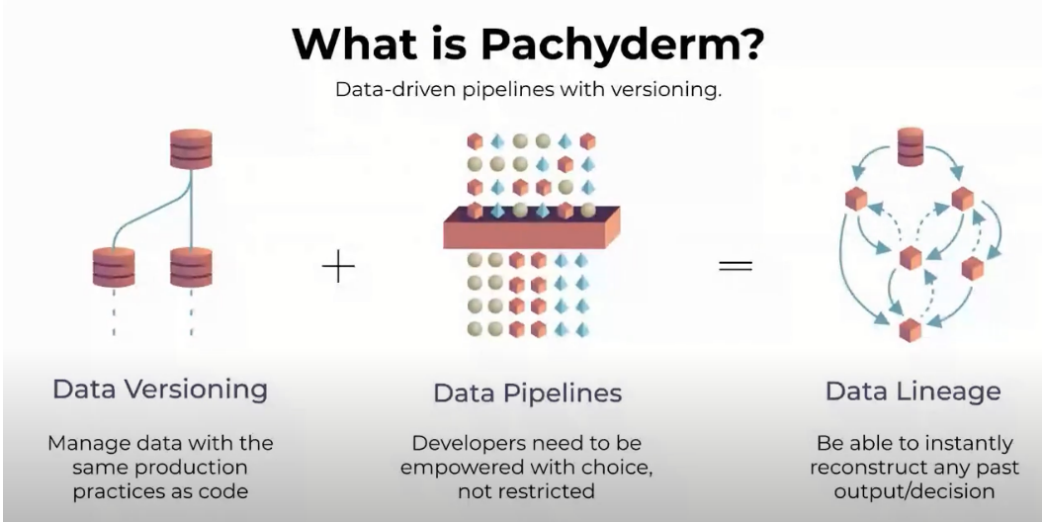
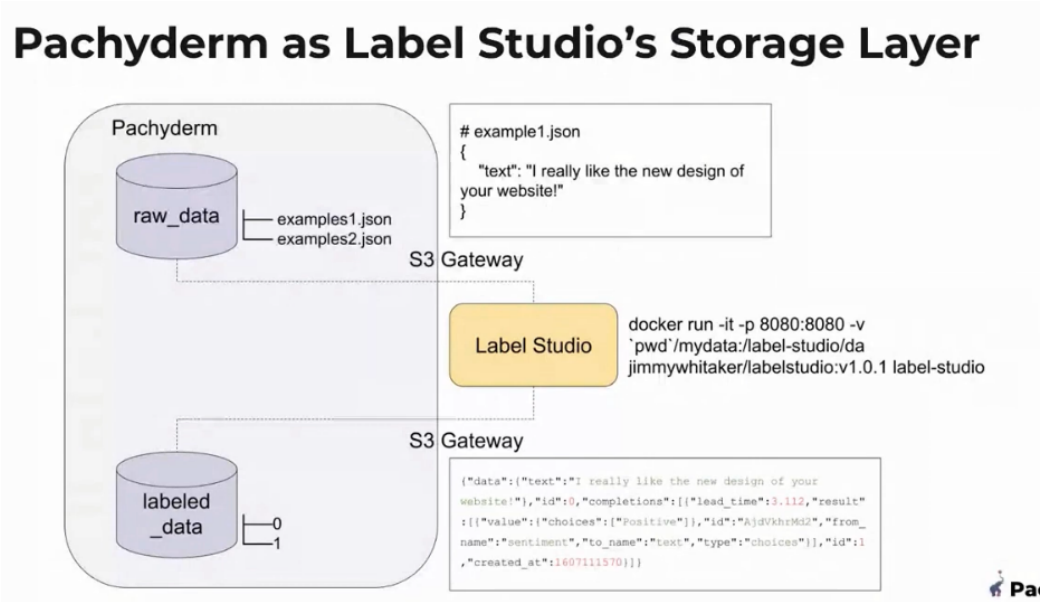
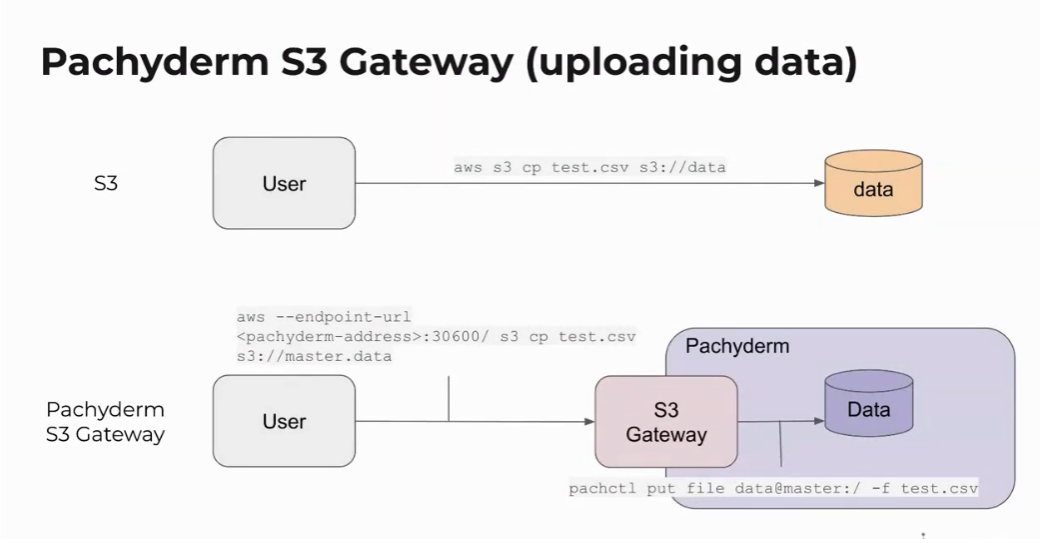
pachyderm技术报告

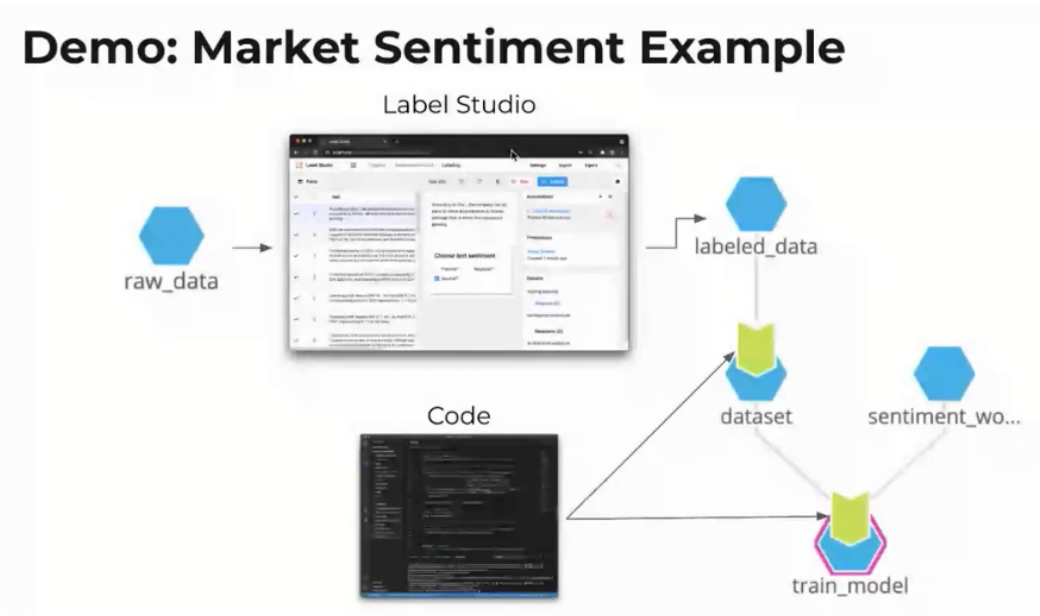
> 背景

mldb的缺点：confluence文档









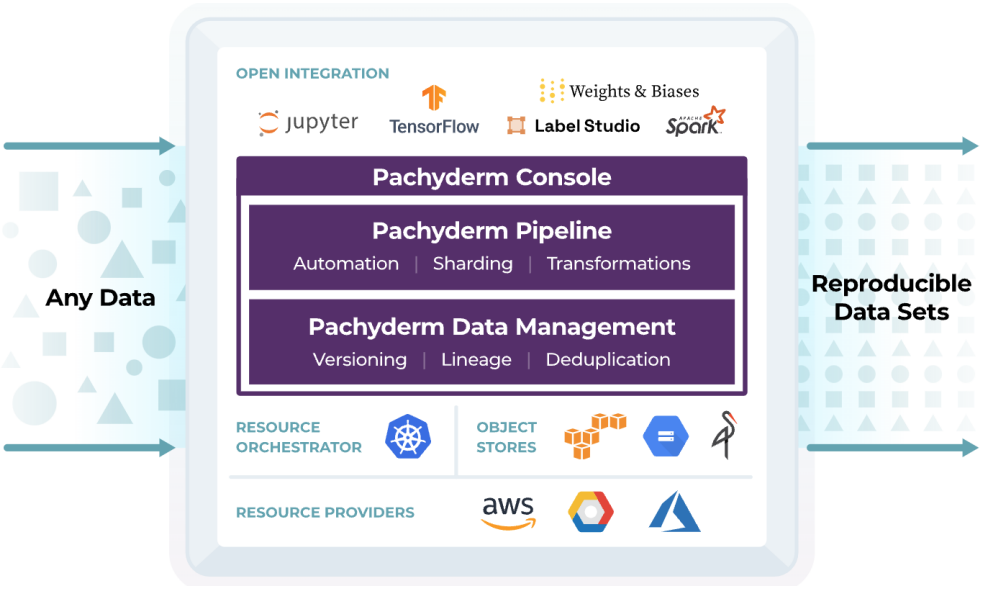
Pachyderm技术特点

数据版本控制和血缘

使用 Pachyderm 构建和扩展容器化、数据驱动的 ML 管道，并通过我们的自动数据版本控制来保证您的数据沿袭。

Pachyderm 是一个企业级的开源数据科学平台，它使可解释、可重复和可扩展的机器学习 (ML) 和人工智能 (AI) 成为现实。Pachyderm 平台将数据版本控制与构建可扩展的端到端 ML/AI 管道的工具结合在一起，同时使用户能够以他们选择的任何语言、框架或工具开发代码。Pachyderm 已被证明是希望使用 ML 和 AI 可靠地解决现实世界问题的团队的理想基础。

架构图：



Pachyderm 是一种以**数据为中心的流水线和版本控制工具**，它允许 ML 和数据科学团队在其组织的数据上实现自动化和协作，同时保持完全的可重复性。

Pachyderm概念

- **Pachyderm File System**(PFS) ：管理 Pachyderm 的数据和版本控制系统。（数据版本控制）**数据版本控制**(History) 使 Pachyderm 用户能够及时返回并查看数据集或存储库在特定时刻的状态。
- **Pachyderm Pipeline System**(PPS) ：使您能够对数据执行各种转换。（数据血缘关系）

Pachyderm数据版本概念

> git区别：

- Repo（存储库）： 高级别的数据对象，一个独立的文件系统。通常，Pachyderm都有自己的存储库
- Commit：commit是pychyderm中数据的不可变快照，对于源数据增删改等操作。与git不同，commit记录了存储库中分支的状态。
- branch：分支是provenance的基本单位。在任何给定时间，它都指向其 repo 在特定提交时的状态，并随着新数据的添加而更新。至关重要的是，一个分支还告诉 Pachyderm 它依赖于什么输入。
- file（文件）：文件时存储库中实际的数据。pychyderm支持任何类型、大小和数量的文件
- Provenance（起源）：表示不同存储库中分支分支之前的关系。了解Pachyderm中所有数据的来源，确保每次提交都包含足够的信息来重建过去。

Repo存储库

存储库是Pachyderm中存储数据的位置，**Pachyderm存储库跟踪对数据的所有更改并创建您可以访问和查看的数据修改历史记录**

与git的区别：Pachyderm提交的历史存储在一个集中的位置，所以不会遇到合并冲突。

Pachyderm 的存储库分为两类'：

1. **用户存储库**

   用户存储库一次一次提交跟踪您的数据。他们进一步分为：

   源存储库

   a. Pachyderm 之外的用户或外部应用程序可以将数据添加到源存储库以进行进一步处理。

   输出存储库

   a. Pachyderm 会在管道末端自动创建一个输出存储库，以便管道将其转换结果写入其中。输出存储库可能用作另一个管道的输入。

2. **系统存储库**

系统存储库保存有关管道的某些辅助信息。它们默认隐藏在大多数命令的输出中。除了输出存储库之外，管道的创建还创建了一个 `spec` 和一个 `meta` 存储库。

a. `spec` 存储库保存管道规范文件
b. `meta` 存储库保存与数据处理相关的元数据（在本文档中也称为“统计信息”）

管道通常管理自己的系统存储库，但如有必要， `edges` 可以使用命名管道的系统存储库 `edges.meta` 以及 `edges.spec` 您通常放置存储库名称的任何位置来引用。删除用户存储库会删除任何关联的系统存储库。

Commit

在 Pachyderm 中，提交是在某个时间点快照并保留存储库中文件和目录状态的原子操作。

与git区别：Pachyderm 提交是集中式的和事务性的。您可以通过 `pachctl start commit` 参考特定存储库运行命令来启动提交。完成对存储库（ `put file` , , ...) 的更改后，您可以通过运行命令 `delete file` 来完成修改。 `pachctl finish commit` 此命令保存您的更改并关闭该存储库的提交，表明数据已准备好由下游管道处理。

commit的三种类型：

- USER：提交是用户更改的结果（ `put file` , `update pipeline` , `delete file` ...)
- AUTO：Pachyderm 的管道是数据驱动的。向数据存储库提交数据可能会触发管道中的下游处理作业。触发作业的输出提交将是类型 `AUTO` 。
- ALIAS：也不 `USER` 是 `AUTO` - `ALIAS` 提交本质上是占位符提交。它们与父提交具有相同的内容，主要用于全局 ID。

支持多个commitID合并成一个commit，也支持删除文件且历史comiit也删除文件（主要针对MLDB删除鉴黄图片需求）

Branch

Pachyderm 分支是一个指向提交的指针，该提交在提交时与新提交一起移动。默认情况下，当您创建存储库时，Pachyderm 不会创建任何分支。 `master` 大多数用户更喜欢通过启动第一次提交并 `master` 在 `put file` 命令中指定分支来创建分支。

当您提交新的更改时， `HEAD` 分支的 将移动到最新的提交。

创建分支cmd

`pachctl create branch <myrepo>@<branchname>` 。可选地，您可以添加 `--head <myrepo>@<master>` 新分支的头部以引用 master 上的头部提交 ）。

File

在Pachyderm存储任何类型文件，包括二进制文件。

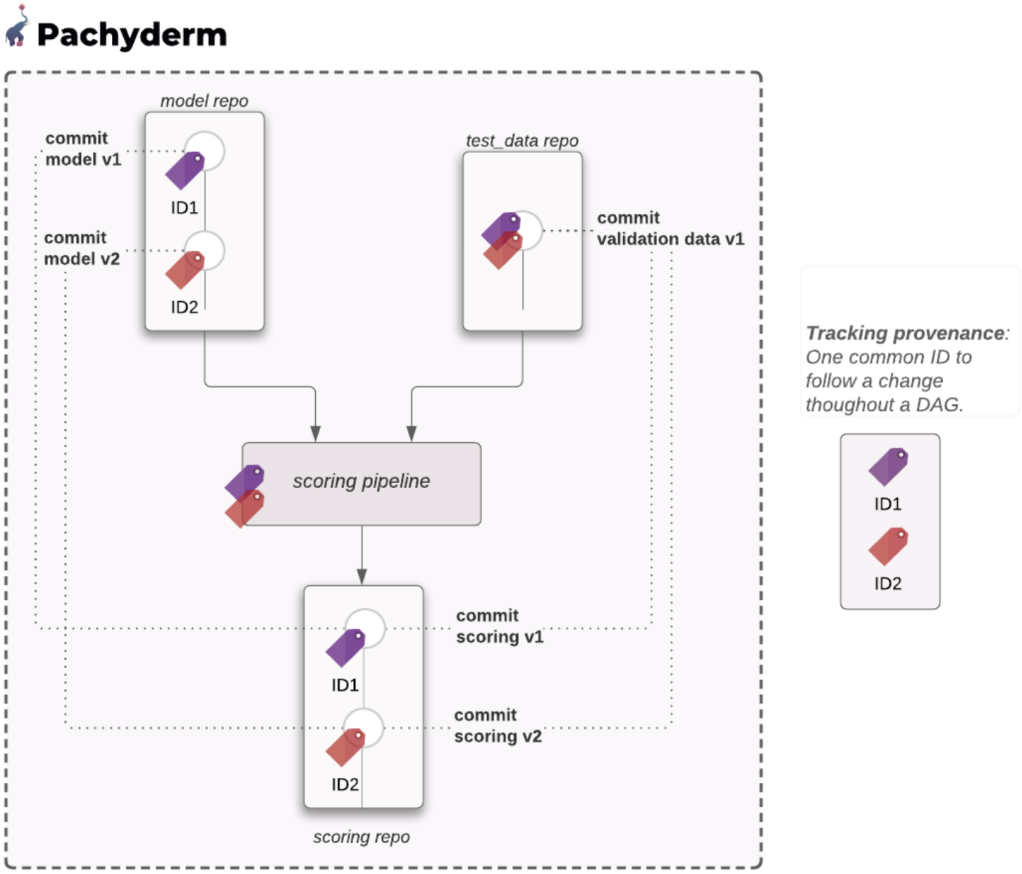
文件处理策略：支持覆盖和附加文件

Probenance（数据来源）

数据出处也称为**数据沿袭**，跟踪数据集之间的依赖关系和关系。它回答了 *“数据从何而来？”的问题。* ，还有 *“一路上数据是如何转换的？”* .

Pachyderm 使其用户能够同时拥有：跟踪其数据的所有修订**并**了解存储在一个存储库中的数据与另一个存储库中的结果之间的联系。

它自动**维护完整的审计跟踪**，允许所有结果完全可重现。



两个输入存储库（`model` 和 `test_data`）为评分管道提供数据。存储库 `model` 不断从外部来源收集模型工件。该管道将来自这两个存储库的数据结合起来，并根据验证数据集对每个模型进行评分。

管道概念（pipeline）

PPF：能够对数据执行各种转换。

- Pipeline（管道）：是等待满足某些条件的作业生成器。最常见的是，这意味着查看一个或多个 Pachyderm 存储库以获取新数据。当新数据到达时，管道会执行一段用户定义的代码来执行操作并处理数据。这些执行中的每一个都称为一个**作业**。
- Job：是管道的单独执行。一项工作可以成功也可以失败。在一项工作中，数据和处理可以分解为单独的工作单元，称为**基准（datum）**。
- Datum：工作中最小的不可分割的工作单元。在一个Job中可以并行处理不同的数据。

Pipeline

管道是一个 Pachyderm 原语，负责从指定源（例如 Pachyderm 存储库）读取数据，根据管道配置对其进行转换，并将结果写入输出存储库。管道订阅一个或多个输入存储库中的分支。每次分支有新的提交时，管道都会执行一个作业，运行您的代码以完成并将结果写入输出存储库中的提交。每个管道都会自动创建一个与管道同名的输出存储库。

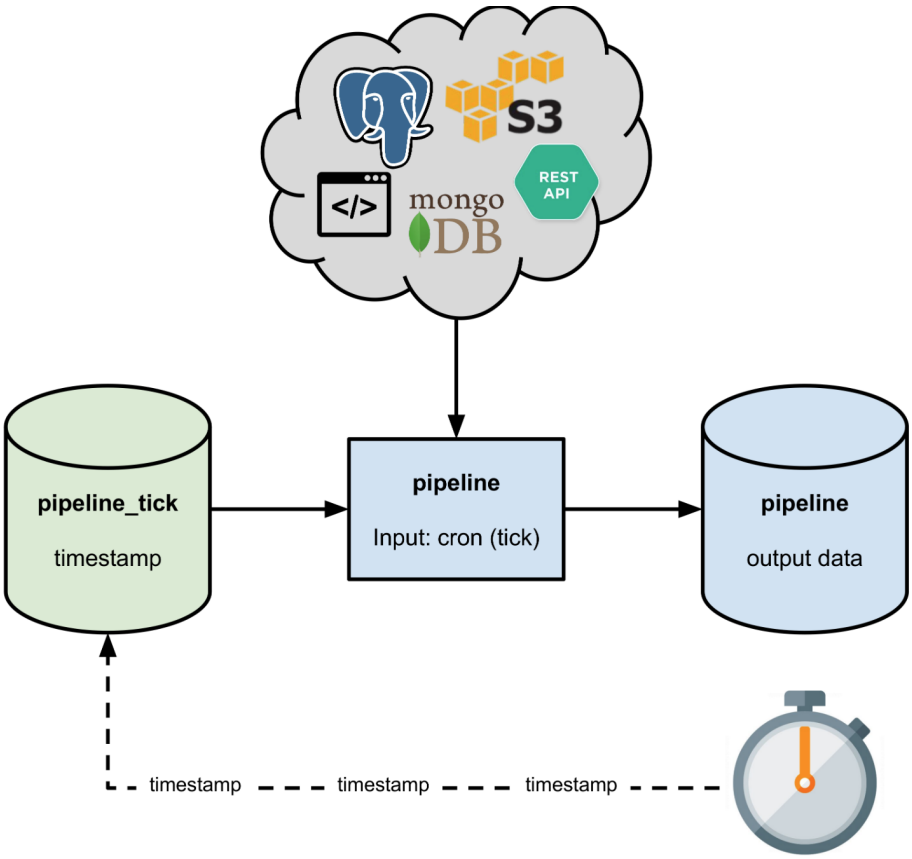
- `name` — 您的数据管道的名称。
- `input` — 您要处理的数据的位置，例如 Pachyderm 存储库。您可以指定多个输入存储库并设置要以各种方式组合的数据。有关详细信息，请参阅Cross and Union、Join、Group。`input` 在该领域中定义的一个非常重要的属性是 `glob` 指定 Pachyderm 如何将数据分解为单独的处理单元（称为基准）的模式。有关详细信息，请参阅基准。
- `transform` — 指定要针对数据运行的代码。该 `transform` 部分必须包含 `image` 定义要运行的 Docker 映像的 `cmd` 字段，以及要执行的容器中的特定代码（例如 Python 脚本）的字段。

```
{
 "pipeline": {
   "name": "wordcount"
 },
 "transform": {
   "image": "wordcount-image",
   "cmd": ["python3", "/my_python_code.py"]
 },
 "input": {
       "pfs": {
           "repo": "data",
           "glob": "/*"
       }
   }
}
```

- Cron：cron 输入使您能够以特定的时间间隔触发管道代码。这种类型的管道对于诸如网络抓取、查询数据库以及您不想等待新数据而是定期触发管道的其他类似操作很有用。
- Service：service是一种特殊类型的管道，它不是执行作业然后等待，而是通过端点永久运行服务数据。例如，您可以提供可查询的 ML 模型或 REST API。服务从 Pachyderm 读取数据，但没有输出存储库。
- Spout：Spout 是一种特殊类型的管道，用于从数据流中提取数据。Spout 可以订阅消息流，例如 Kafka 或 Amazon SQS，并在收到消息时摄取数据。spout 没有输入 repo。

Cron

当输入存储库中出现新更改时，Pachyderm 会触发管道。但是，如果您想根据时间而不是在输入数据到达时触发管道，您可以使用 Pachyderm 内置的 cron 输入类型安排此类管道定期运行。



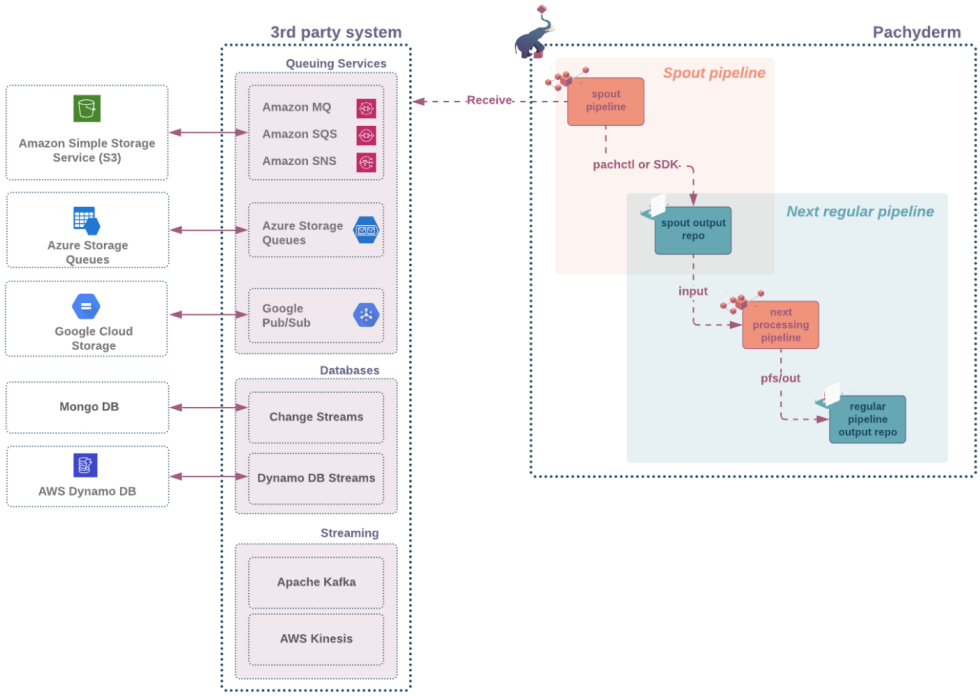
Service

服务是一种特殊类型的管道，它不处理数据，但提供将数据公开给外界的能力。例如，您可以使用服务将机器学习模型作为 API 提供，该 API 具有最新版本的数据。

```
{
    "pipeline": {
      "name": "notebook"
    },
    "input": {
        "pfs": {
            "glob": "/",
            "repo": "input"
        }
    },
    "service": {
        "external_port": 30888,
        "internal_port": 8888
    },
    "transform": {
        "cmd": [
            "start-notebook.sh"
        ],
        "image": "jupyter/datascience-notebook"
    }
}
```

Spout

Spout 是一种管道类型，它**从外部源**（消息队列、数据库事务日志、事件通知.....）摄取流数据.



要创建一个 spout 管道

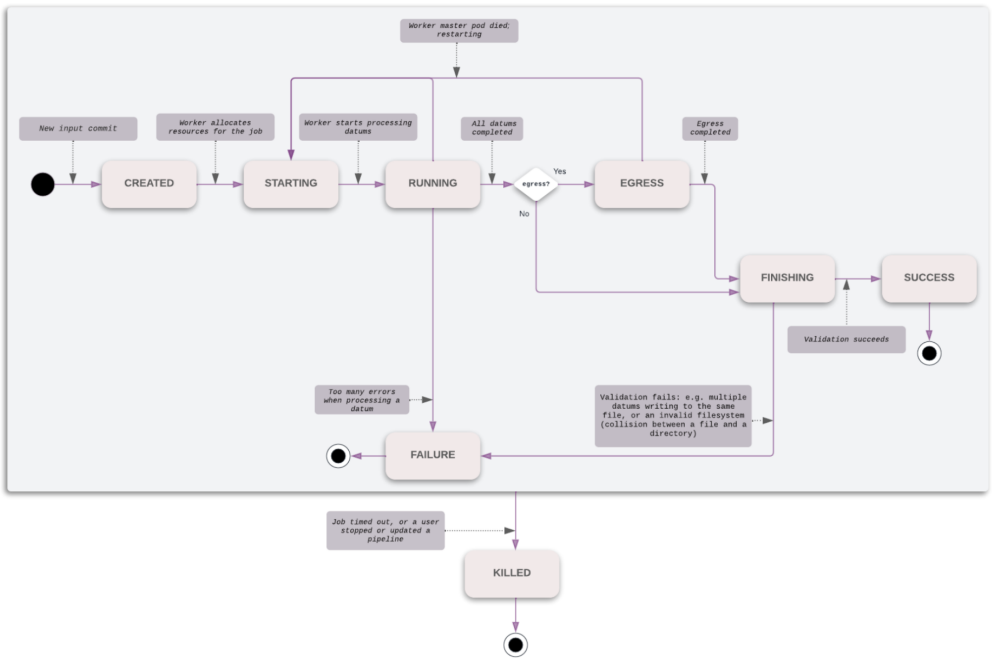
- 流数据的来源。

- 一个带有 spout 代码的 Docker 容器，用于连接、读取、转换并将数据从数据源推送到输出存储库。
- 使用您的容器的 spout 管道规范文件。

```
{
  "pipeline": {
    "name": "my-spout"
  },
    "spout": {
  },
  "transform": {
    "cmd": [ "go", "run", "./main.go" ],
    "image": "myaccount/myimage:0.1",
    "env": {
        "HOST": "kafkahost",
        "TOPIC": "mytopic",
        "PORT": "9092"
    }
  }
}
```

Job

Job是在输入存储库中检测到新数据时触发的管道的执行。当有新的数据写入，pipeline会出发新的job去执行。下图为job运行的状态图。




```
{
    "job": {
        "pipeline": {
        "name": "edges"
        },
        "id": "fd9454d06d8e4fa38a75c8cd20b39538"
    },
    "pipeline_version": "1",
    "output_commit": {
```

```
        "branch": {
        "repo": {
            "name": "edges",
            "type": "user"
        },
        "name": "master"
        },
        "id": "fd9454d06d8e4fa38a75c8cd20b39538"
    },
    "data_processed": "2",
    "data_skipped": "1",
    "data_total": "3",
    "stats": {
        "download_time": "0.113263653s",
        "process_time": "1.020472976s",
        "upload_time": "0.010323995s",
        "download_bytes": "185424",
        "upload_bytes": "114041"
    },
    "state": "JOB_SUCCESS",
    "created": "2021-08-02T20:13:10.461841493Z",
    "started": "2021-08-02T20:13:32.870023561Z",
    "finished": "2021-08-02T20:13:38.691891860Z",
    "details": {
        "transform": {
        "image": "pachyderm/opencv",
        "cmd": [
            "python3",
            "/edges.py"
        ]
        },
        "input": {
        "pfs": {
            "name": "images",
            "repo": "images",
            "repo_type": "user",
            "branch": "master",
            "commit": "fd9454d06d8e4fa38a75c8cd20b39538",
            "glob": "/*"
        }
        },
        "salt": "27bbe39ccae54cc2976e3f960a2e1f94",
        "datum_tries": "3"
    }
}
```


Datum

- 定义：数据是Job**中最小的不可分割的计算单元。**一项job可以有一个、多个或没有datum。每个数据都是独立处理的，用户代码在其中一个管道工作 pod 上执行一次。然后

将所有数据输出的文件组合在一起以创建最终的输出提交。

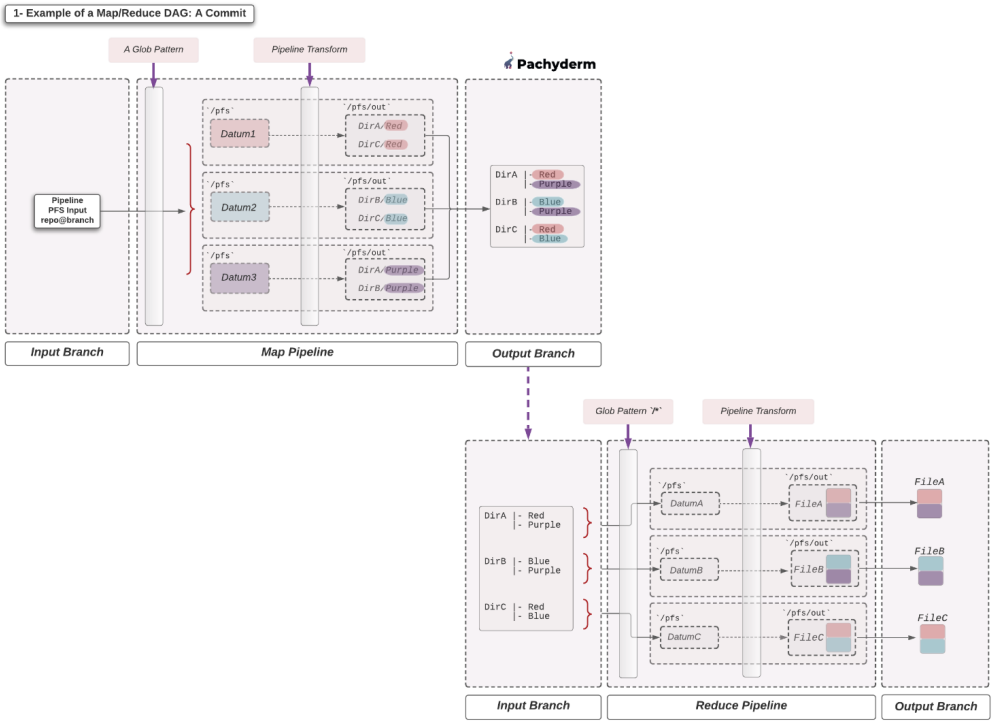
- 数据分布：将Datum视为**划分输入数据**和**分配处理工作负载**的一种方式。它们有助于优化您的管道性能。
- PFS输入和Glob模式：监听提交的分支，Glob模式用来确定输入数据是如何区分的。一个pipeline可以有一个或者多个repo输入。在多个repo的情况下，Pachyderm会将多个repos聚合在一起
- 聚合Datum：多个repos聚合
  - Cross（混合）：输入为多个存储库。来自一个存储库的每个数据都与来自另一个存储库的每个数据相结合。
  - Group：可以采用一个或多个存储库，并一次处理与特定文件路径模式匹配的所有数据。Pachyderm 的组类似于数据库*group-by*，但它仅匹配文件路径，而不匹配文件的内容。
  - Union：联合输入可以采用多个存储库并独立处理每个输入中的所有数据。管道以未定义的顺序处理数据，输出存储库包括来自所有输入源的结果。
  - Joins：连接输入使您能够连接存储在不同 Pachyderm 存储库中的文件并匹配特定的文件路径模式。Pachyderm 支持类似于数据库的*内连接*和*外连接*操作的连接，尽管它们只匹配文件路径，而不是实际文件内容。
- Glob patterm（datum数据的划分）
  - 定义数据如何在工作人员之间传播是分布式计算最重要的方面之一。
  - ‘/’将整个存储库表示为单个数据，也就是一个datum
  - ‘/* ‘当前目录下的每个文件/目录作为单独的datum
  - ‘/*/*’ 每个子目录中的每个文件或目录作为单独的datum
  - /** 所有目录和子目录中的每个文件作为单独的数据处理
- Unio输入

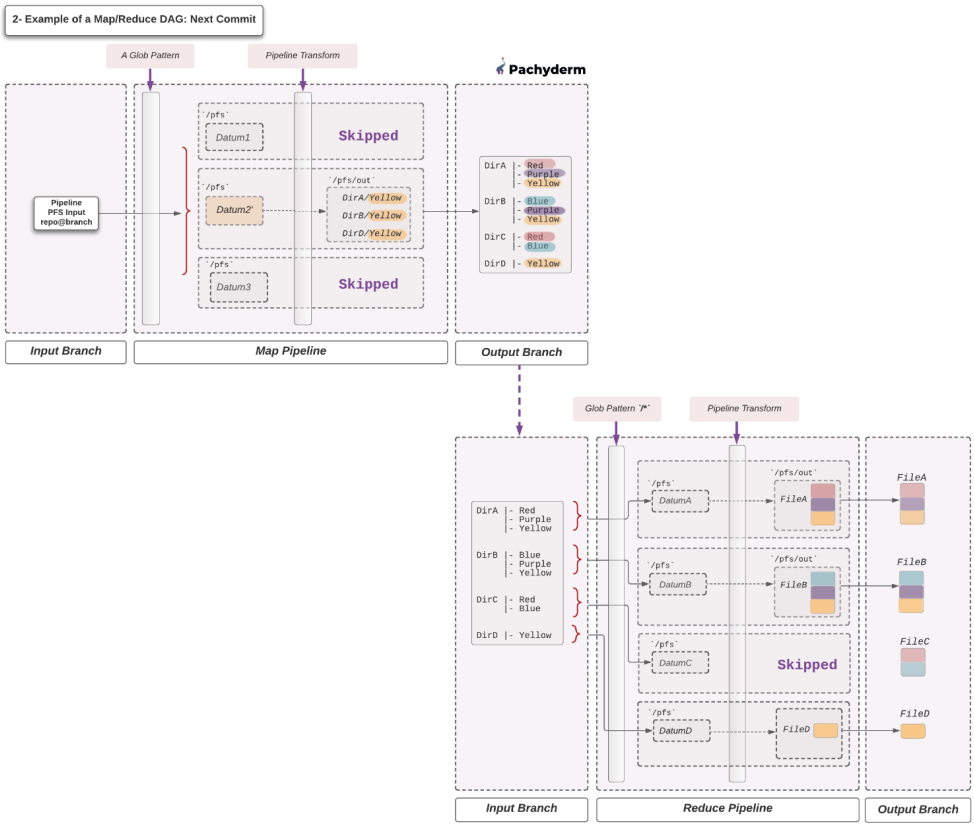
```
"input": {
    "union": [
        {
            "pfs": {
                "glob": "/*",
                "repo": "A"
            }
        },
        {
            "pfs": {
                "glob": "/*",
                "repo": "B"
            }
        }
    ]
}
// 1 个datum ,A and B have 3 files ,pfs中增加name指定文件名，使其在同一个目录下
/pfs/A/1.txt
/pfs/A/2.txt
/pfs/A/3.txt
/pfs/B/4.txt
/pfs/B/5.txt
/pfs/B/6.txt
```

- Cross输入

```
3个datum
Run 1: /pfs/A/1.txt
       /pfs/B/4.txt
Run 2: /pfs/A/1.txt
       /pfs/B/5.txt
...
Run 9: /pfs/A/3.txt
       /pfs/B/6.txt
```

管道数据处理流程（datum）





datum数据处理状态

datum基准统计

Pachyderm 存储有关管道处理的每个数据的信息，包括时间信息、大小信息和 `/pfs` 快照。

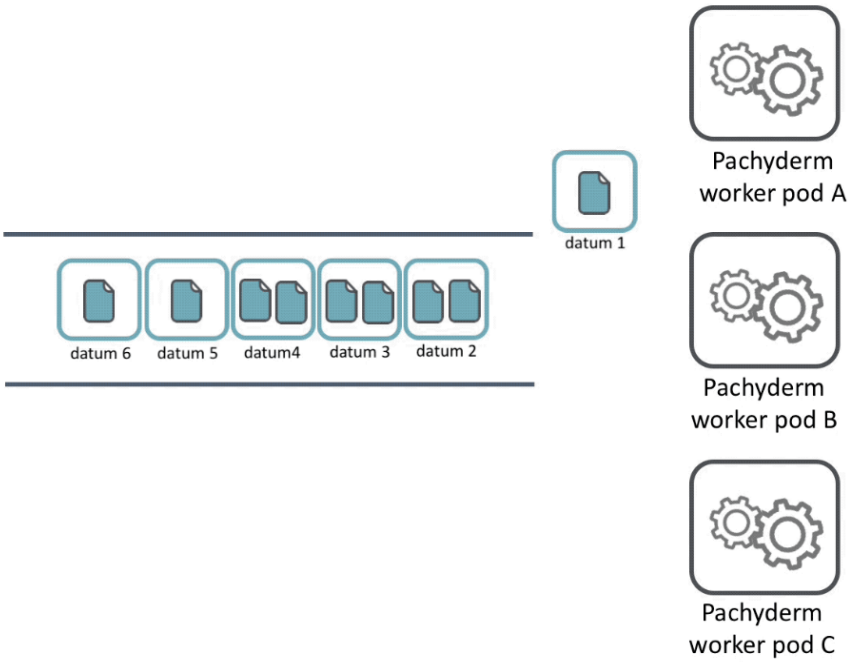
```
pachctl get file edges.meta@master:/meta/002f991aa9db9f0c44a92a30dff8ab22e788f86cc851bec80d5a74e05ad12868/meta | jq

{
  "job": {
    "pipeline": {
      "name": "edges"
    },
    "id": "efca9595bdde4c0ba46a444a5877fdfe"
  },
  "inputs": [
    {
      "fileInfo": {
        ...
    }
  ],
  "hash": "28e6675faba53383ac84b899d853bb0781c6b13a90686758ce5b3644af28cb62f763",
  "stats": {
    "downloadTime": "0.103591200s",
    "processTime": "0.374824700s",
    "uploadTime": "0.001807800s",
    "downloadBytes": "80588",
    "uploadBytes": "38046"
  },
  "index": "1"
}
```

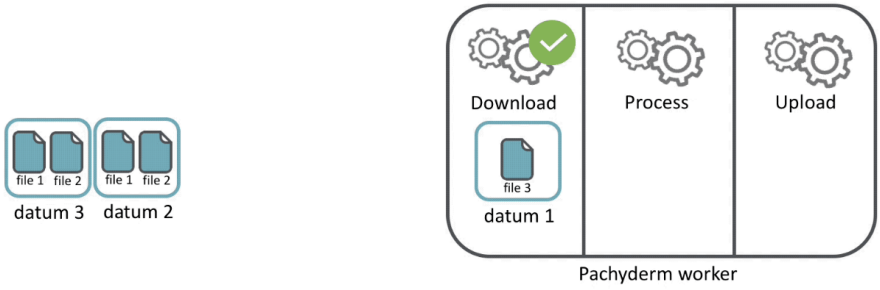
全局ID

延迟处理数据

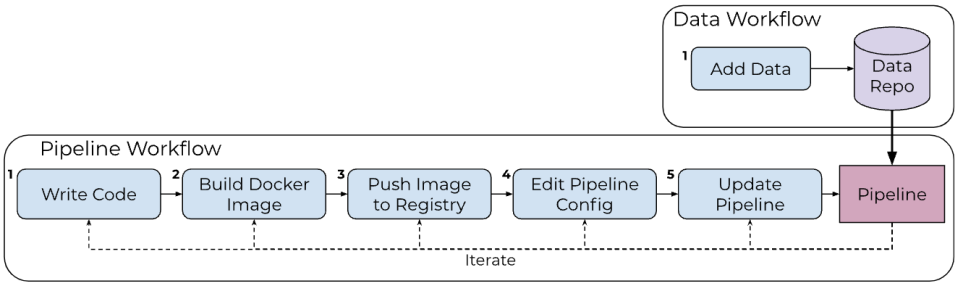
分布式计算

当您创建管道时，Pachyderm 会启动在集群中持续运行的工作 pod，等待新数据可用于处理。您可以通过设置来更改此行为 `"autoscaling" :true` 。因此，您不需要为每个新作业重新创建和安排工作人员。

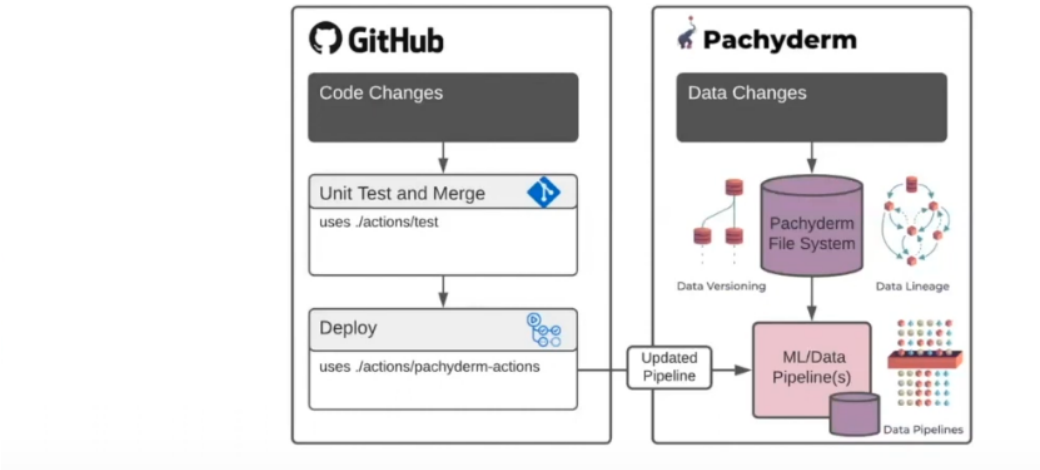




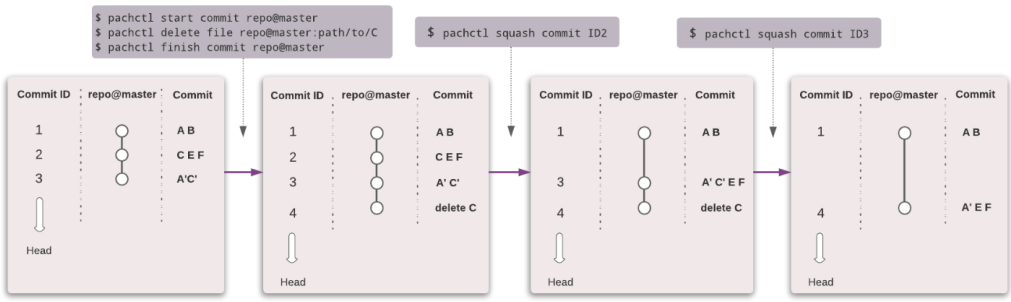
pipeline工作流程



DevOps + MLOps

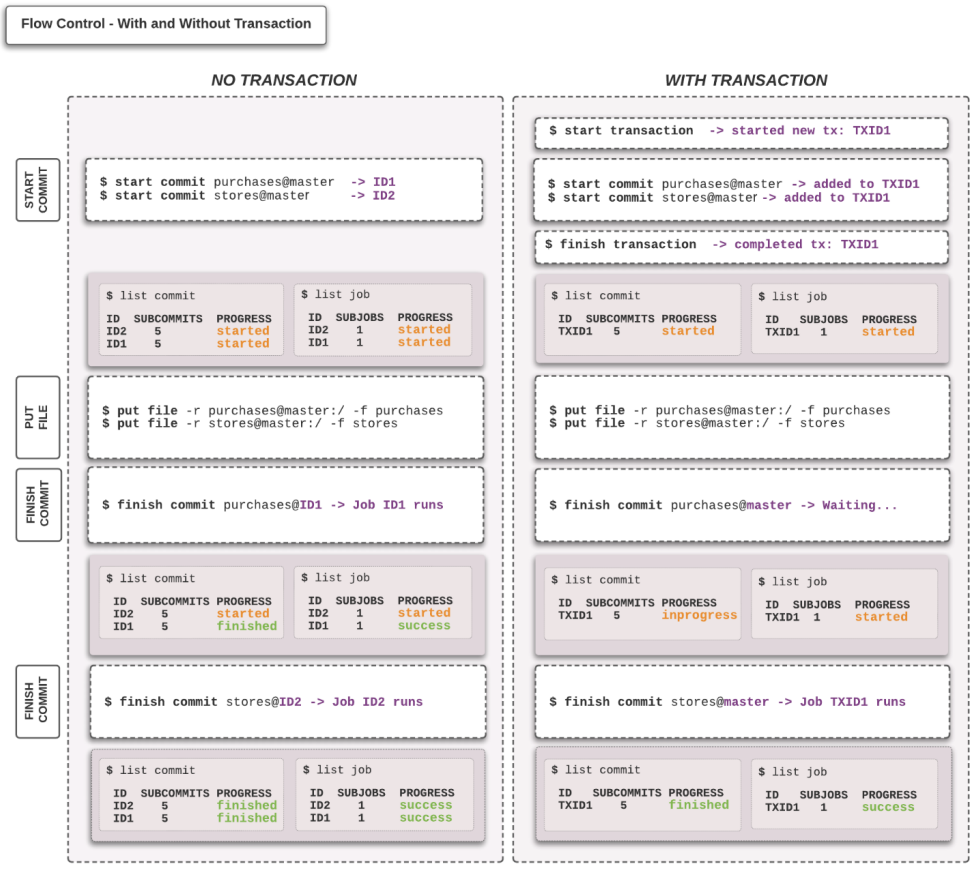


删除提交历史中的文件

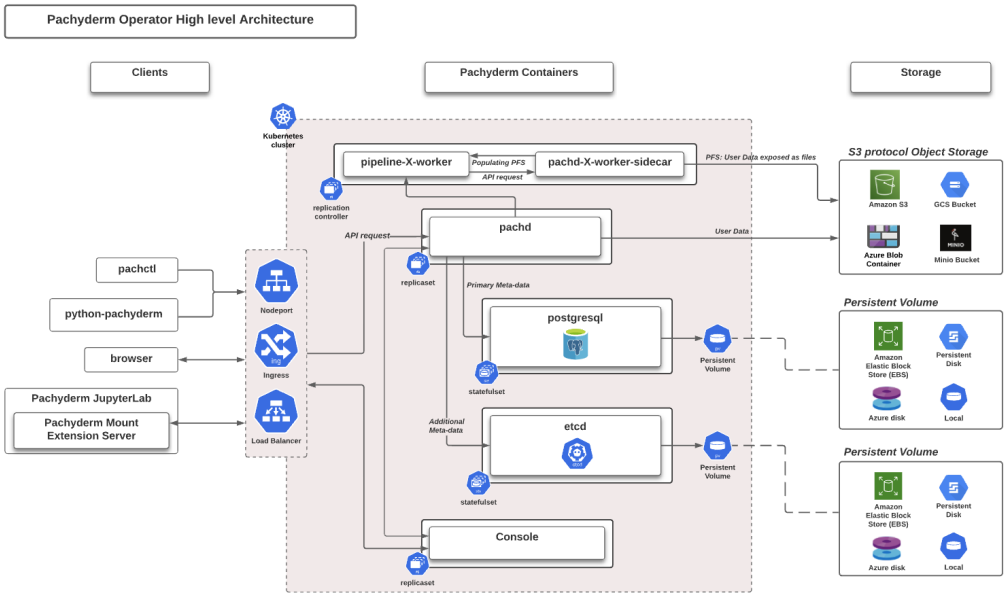


事务

使用事务，在一个作业运行中同时运行多个Pachyderm命令



Pachyderm技术架构



Pachyderm版本控制

Pachyderm数据血缘