IDUG VIRTUAL

2021 NA **Db2** Tech Conference

Pavel Sustr, **Michael Wang**, IBM Canada Lab

@pavel_sustr

Db2 Historical Monitoring

Db2 for Linux, UNIX, Windows

Agenda

- Business Case
- Architectural Overview
- Installation
- Reports
- Demo
- Submit Changes

Business Case

- Until now, Db2 has provided limited ability to perform root cause analysis for one-time problems or issues whose timeframe of occurrence could not be predicted reliably (i. e. not reproducible on demand)
- Usual approach involves reviewing Db2 diagnostic logs, Db2 administration notification log, and other diagnostic files if present (e. g. dump/stack/trap files, etc.)
- This approach would often be insufficient, ***especially with hangs or temporary performance problems***
- In order to determine root cause, problem would often need to be reproduced again with Db2 trace or other techniques enabled

Solution: Db2 Historical Monitoring (`db2histmon`)

- Improve first occurrence problem determination capabilities by continuous **lightweight** collection of detailed monitoring of Db2 workload processing
- Not intended to provide operational history
  - Although could potentially be used in this way, and could be expanded in future to address this need, the initial focus is on the needs of IBM Db2 Support
- Not a replacement for enterprise monitoring applications such as IBM Db2 Data Management Console (a.k.a. IBM Data Server Manager)
  - More intended for the collection of mid and low-level diagnostic data
- Not a replacement for low level problem determination techniques such as Db2 trace, trap files, dump files, etc.

High-Level Architectural Overview

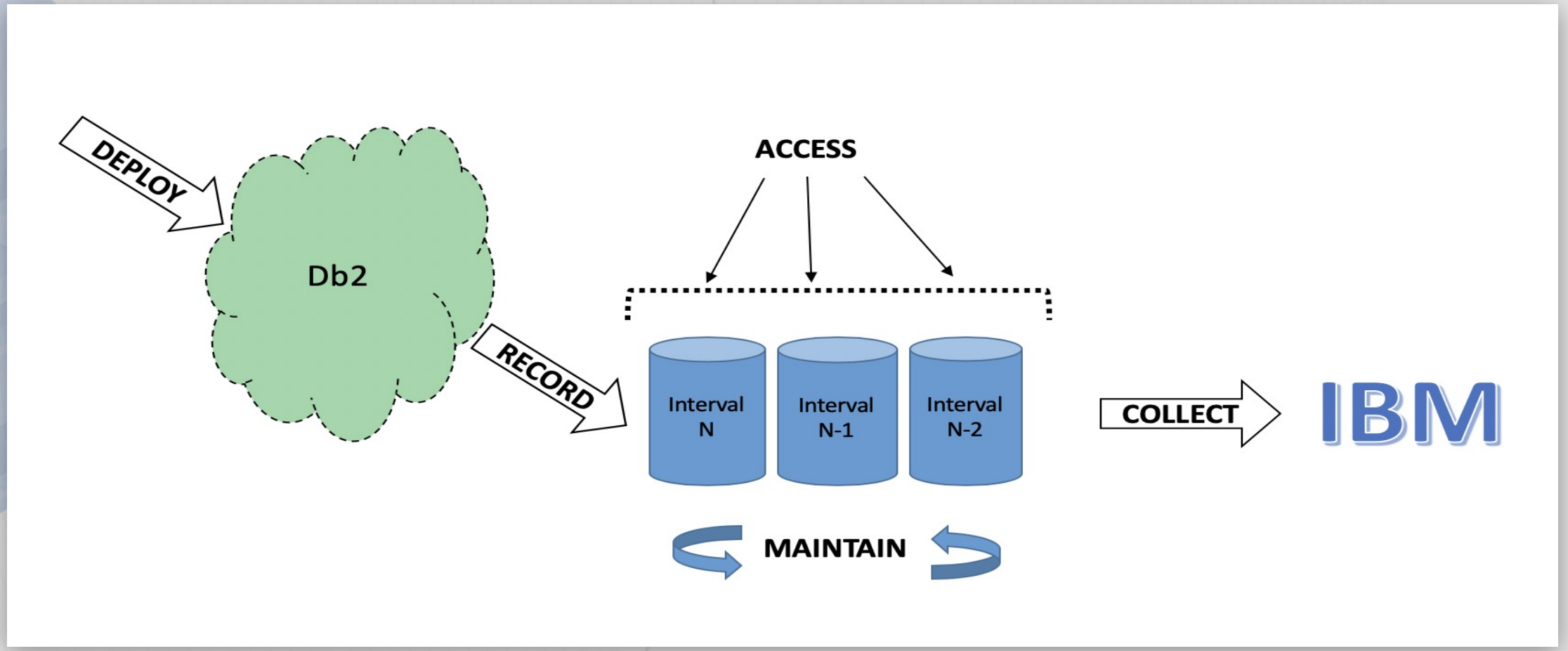
- Set of (human-readable) Python scripts used to set up and maintain series of Administrative Task Scheduler (ATS) tasks, and parsers to process and visualize data
- ATS tasks responsible for collecting (and archiving) lightweight operational data by utilizing `MON_GET_*` functions or operating system calls
- Set of C++ functions required to interface with the operating system and handle operations that SQL cannot
- Information gathered stored in text files independent of Db2 instance
  - Can be easily loaded into new database for SQL queries
  - Reporting tools and parsers available
- Resides in public GitHub repository, your input and contribution are encouraged

Why Current Design?

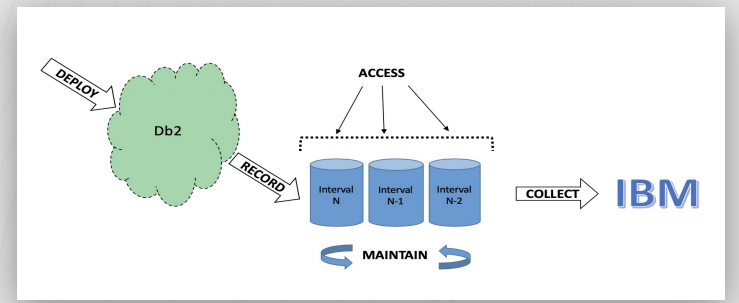
- Current design allows for quick creation of a robust framework and easy modification
- ATS is the default choice for automated database tasks (reorg etc.)
- `MON_GET_*` infrastructure is documented, allows to control data volume, output data can be easily parsed and loaded into tables
- Filesystem storage uses minimal database resources, is available all the time regardless of the instance status, and provides flexibility for placement and location
- Python is suitable for easy data parsing and processing

DISCLAIMER: Design may change at any time depending on user input and field experience.

Workflow Overview



Workflow Phases 1/2



Deploy

- Deploy default historical monitoring to database with default configuration for storage location and maximum consumption, allowing to easily customize these values for each database

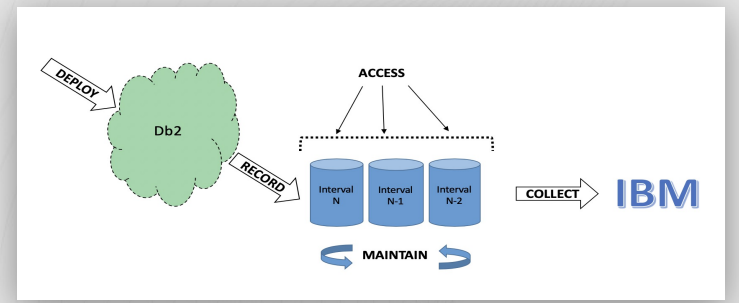
Record

- Capture information relevant to the target scenario at frequent intervals

Access

- Make monitored data available for immediate viewing
- Make monitored data available for SQL access
- Make MONREPORT-like reports from monitored data

Workflow Phases 2/2



Maintain

- Historical monitoring stays within defined limit for storage consumption
- When storage limit is reached, monitoring data wraps to beginning
- Old data is deleted (archiving could be configured)

Collect

- Explicitly collect the existing monitored data on demand for shipment to IBM Support (e. g. user can request data capture at time of incident)

Scenarios of Interest (First Drop)

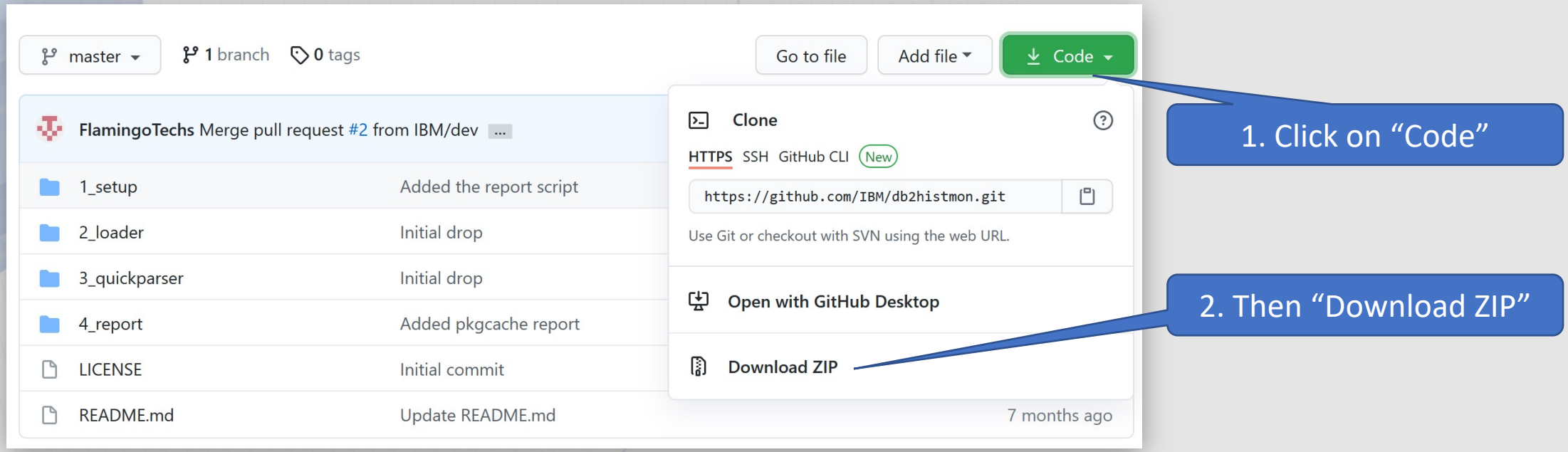
- Contention (latch, lock, log file, etc.) and high-impact SQL operations
- Remote (MPP) failure scenarios
- Memory exhaustion and paging
- HADR congestion
- Additional broad diagnostic data for general usefulness

Pre-requisites

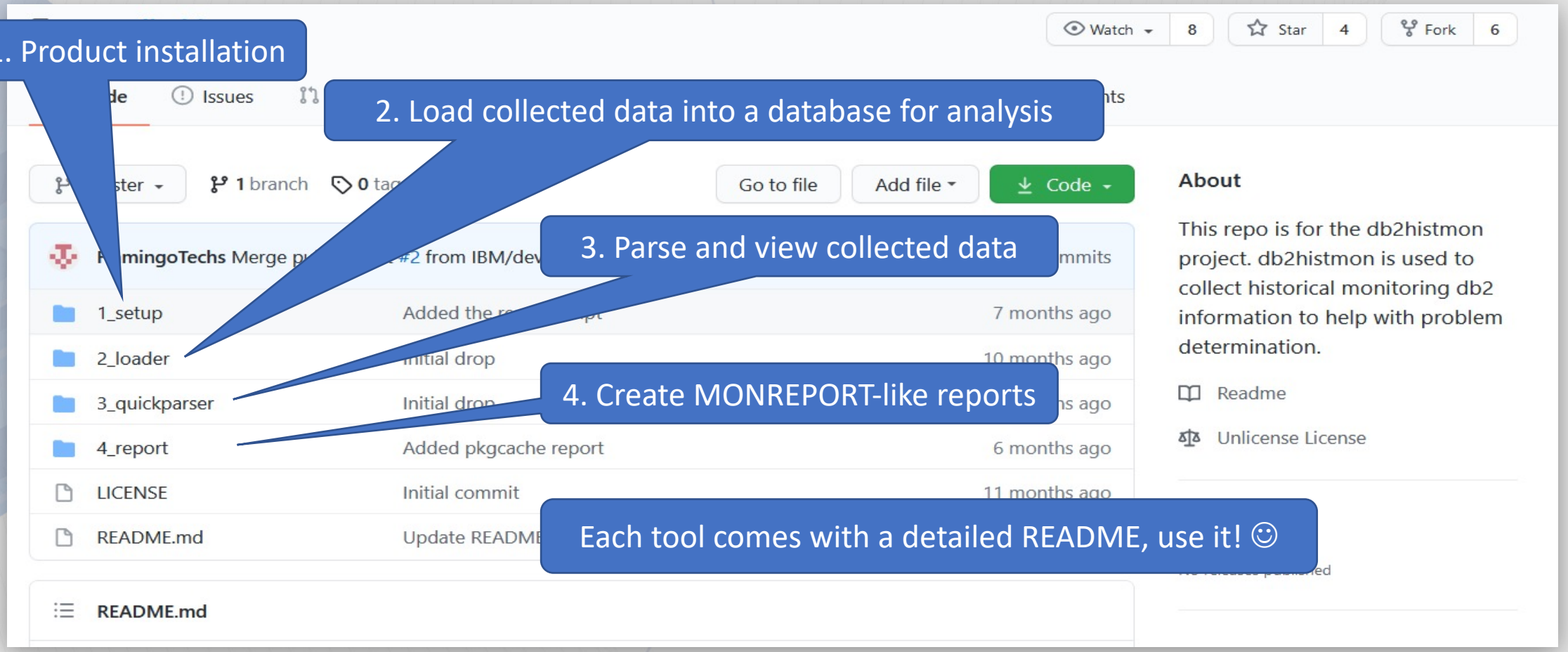
- Db2 for Linux, UNIX, Windows
  - Most modern releases are supported, e. g. 10.1, 10.5, 11.1, 11.5
  - Can be enabled on any platform: Linux, UNIX, or Windows
- Python
  - Version 3 or newer (version 2 will NOT work)
  - `ibm_db2` module for Db2 connectivity
  - `pandas` module for reporting feature
- C++ compiler
  - E. g. GNU GCC on Unix/Linux or Microsoft Visual Studio on Windows
  - Needed to compile external C++ UDFs used by the monitoring framework
- The **System DBADM** authority

Get Db2 Historical Monitoring

- If you have an active GitHub setup, simply clone the package:
  - `git clone https://github.com/IBM/db2histmon.git`
- Alternatively, download from https://github.com/IBM/db2histmon:



Quick Anatomy



Enablement, Modification, Disablement

- Installation

```
pip3 install ibm_db
cd db2histmon/1_setup
python3 setup.py –h
python3 setup.py <db>
```

- Configuration change

```
python3 setup.py <db> –uc
python3 setup.py <db> –ut
```

- Uninstallation

```
python3 setup.py <db> –c
```

```
$ python3 setup.py -h
usage: setup.py [-h] [-un] [-pw] [-c] [-uc] [-ut] [-
bp] [-cp] [-ap] [-ms]
                [-acmd] [-aext] [-lvl]
                database

Setup IBMHIST monitoring program. View README for more
information.

positional arguments:
  database              name of database

optional arguments:
 -h, --help            show this help message and exit
 -un , --username      username used to connect
 -pw , --password      password used to connect
 -c, --cleanup         will drop the IBMHIST schema

<…snip…>
```

Monitoring Status

```
db2 "SELECT substr(T1.name, 1, 10)
        as name,
        T1.taskid,
        T2.sqlcode,
        T2.status,
        T2.invocation,
        T2.end_time
FROM
        SYSTOOLS.ADMIN_TASK_LIST T1
LEFT JOIN
        SYSTOOLS.ADMIN_TASK_STATUS T2
ON T1.taskid = T2.taskid"
```

```
NAME       TASKID SQLC STATUS    IN END_TIME
---------- ------ ---- --------  -- ----------
MON_GET_UN     85    0 COMPLETE   1 2021-04-26-10.42
MON_GET_AC     84    0 COMPLETE   1 2021-04-26-10.42
VMSTAT        111    0 COMPLETE   1 2021-04-26-10.42
MON_GET_CO     83    0 COMPLETE   1 2021-04-26-10.42
MON_GET_SE    104    0 COMPLETE   1 2021-04-26-10.42
MON_GET_RE    103    0 COMPLETE   1 2021-04-26-10.42
MON_GET_LA     99    0 COMPLETE   1 2021-04-26-10.42
NETSTAT       113    0 COMPLETE   1 2021-04-26-10.42
IOSTAT        112    0 COMPLETE   1 2021-04-26-10.42
MON_GET_EX     96    0 COMPLETE   1 2021-04-26-10.42
MON_GET_EX     88    0 COMPLETE   1 2021-04-26-10.42
ENV_GET_SY     81    0 COMPLETE   1 2021-04-26-10.42
MON_GET_PA    101    0 COMPLETE   1 2021-04-26-10.42
MON_GET_ME     89    0 COMPLETE   1 2021-04-26-10.42
MON_GET_UT    107    0 COMPLETE   1 2021-04-26-10.42
MON_GET_TR     91    0 COMPLETE   1 2021-04-26-10.42
NETSTAT       113    0 COMPLETE   2 2021-04-26-10.43
<…snip…>
```

Output

Database Changes

- New schema `IBMHIST`
- New tables `TAB_CONFIG`, `TAB_DIRS`, `TAB_TASKS`, `TAB_ERRS`
- New ATS tasks

Data

- Structured text files in `$DIAGPATH/IBMHIST_<db>`
- Every hour goes to a separate directory
- Archive in `$DIAGPATH/IBMHIST_<db>_archive`
- Configurable by `setup.py -cp <newpath> -uc`

Frequency

- Minimum: 1 min, maximum: 60 min, default: varies (mostly 1 min though)
- Configurable by editing `task_details.json` *(see subsequent slides)*

What Is Collected?

- Two types of data: SQL and SYS *(system, things SQL cannot achieve)*

```
$ grep "collection_name" 1_setup/task_details.json | awk '{print $2}'
"ENV_GET_SYSTEM_RESOURCES",      "MON_GET_MEMORY_POOL",             "MON_GET_PKG_CACHE_STMT",
"ENV_CF_SYS_RESOURCES",          "MON_GET_MEMORY_SET",              "MON_GET_REBALANCE_STATUS",
"MON_GET_AGENT",                 "MON_GET_TRANSACTION_LOG",         "MON_GET_SERVERLIST",
"MON_GET_CONNECTION",            "DB_GET_CFG",                      "MON_GET_TABLE",
"MON_GET_ACTIVITY",              "DBMCFG",                          "MON_GET_TABLESPACE",
"MON_GET_UNIT_OF_WORK",          "ENV_INST_INFO",                   "MON_GET_UTILITY",
"MON_CURRENT_SQL",               "ENV_GET_REG_VARIABLES",           "MON_GET_WORKLOAD",
"MON_GET_APPL_LOCKWAIT",         "MON_GET_EXTENT_MOVEMENT_STATUS",  "MON_GET_FCM",
"MON_GET_CF_CMD",                "MON_GET_GROUP_BUFFERPOOL",        "MON_GET_DATABASE",
"MON_GET_CF",                    "MON_GET_INDEX",                   "VMSTAT",
"MON_GET_CF_WAIT_TIME",          "MON_GET_LATCH",                   "IOSTAT",
"MON_GET_EXTENDED_LATCH_WAIT",   "MON_GET_BUFFERPOOL",              "NETSTAT",
"MON_GET_HADR",                  "MON_GET_PAGE_ACCESS_INFO",
```

- SQL output is comma-delimited (easily parsable), SYS output is raw

Retention

- Primary path `$DIAGPATH/IBMHIST_<db>` only contains data for current hour
- Every hour, a scheduled task moves data for past hour to archive at `$DIAGPATH/IBMHIST_<db>_archive`, and compresses the data
- If size of “primary + archive” exceeds `IBMHIST.TAB_CONFIG.MAXSIZE` (default 1 GB), oldest data in archive is physically removed
- Configurable by `setup.py -ms <newsize> -uc`

Tasks: tasks_details.json

- All tasks defined in 1_setup/tasks_details.json
- Sample task definition:

```
{
   "collection_name": "MON_GET_MEMORY_POOL",
   "collection_class": "SQL",
   "collection_command": "SELECT CURRENT TIMESTAMP AS COLLECTION_TIME, T.*
                          FROM TABLE (MON_GET_MEMORY_POOL( NULL, NULL, -2) ) T",
   "collection_freq": "* * * * *",
   "collection_level": "1",
   "collection_condition": "",
   "loader_join_columns": "MEMORY_POOL_ID, MEMBER",
   "loader_diff_exempt_columns": "MEMORY_POOL_ID, APPLICATION_HANDLE, EDU_ID",
   "quickparse_summary_columns": "ALL"
}
```

Task Customizations

1. Add new tasks by modifying `1_setup/tasks_details.json`
   - **collection_name** – arbitrary name describing the task
   - **collection_class** – **SQL** for SQL queries, or **SYS** for operating system calls
   - **collection_command** – command to execute
   - **collection_freq** – collection frequency in `crontab` format
   - **collection_level** – level of deployment, see `setup.py -lvl`
   - **collection_condition** – empty, or one of [HADR|PURESCALE|WINDOWS]
   - **loader_join_columns** and **loader_diff_exempt_columns** – leave empty
   - **quickparse_summary_columns** – set to **ALL**
2. Load new configuration by running `1_setup/setup.py -ut`

Reporting/Data Parsing Techniques

- Three shipped techniques to parse collected data
- Mainly intended for IBM Service, but user testing and input welcome
- By default, only current hour's data (in primary path) is processed
- If needed, un-compress past data in archive and point the tools there

a) **Loader**
   - Loads output data to Db2 database

b) **Quick Parser**
   - Formats raw data in user-friendly way

c) **Report Script**
   - Generates reports like MONREPORT

Loader

- Located in `2_loader/loader.py`, allows to load existing data into Db2 database
  - User provides existing database name
  - Tool creates table space `HISTMON`, schema `IBMHIST`
  - Data loaded into `IBMHIST.ENV_*` and `IBMHIST.MON_GET_*` tables
  - Deltas into `IBMHIST.ENV*_DELTA` and `IBMHIST.MON*_DELTA` tables
- User can use SQL queries to examine the data as usual
- Usage
  - `cd db2histmon/2_loader`
  - `python3 loader.py -h`
  - `python3 loader.py -d MONDB -sourcePath ~/sqllib/db2dump/IBMHIST_SAMPLE`

Loader – Execution Example

```
$ loader.py -d MONDB -sourcePath ~/sqllib/db2dump/IBMHIST_SAMPLE
Connecting to database: MONDB
Drop table space HISTMON
Create table space HISTMON
Loading tasks from task_details.json ...
Creating the data table: IBMHIST.ENV_GET_SYSTEM_RESOURCES
Loading data into IBMHIST.ENV_GET_SYSTEM_RESOURCES

<…skipping…>

Creating the delta data table: IBMHIST.MON_GET_FCM_DELTA
Loading data into IBMHIST.MON_GET_FCM_DELTA
Creating the data table: IBMHIST.MON_GET_DATABASE
Loading data into IBMHIST.MON_GET_DATABASE
Creating the delta data table: IBMHIST.MON_GET_DATABASE_DELTA
Loading data into IBMHIST.MON_GET_DATABASE_DELTA
Closing connection ...
```

Loader – Sample Queries 1

- Display all available tables to query:
  - `select TABNAME from SYSIBMADM.ADMINTABINFO where tabschema='IBMHIST'`
- Determine if time is spent on database server vs client:
  - `select collection_time, member, avg(client_idle_wait_time) client_idle_wait_time, avg(total_rqst_time) total_rqst_time from ibmhist.mon_get_connection_delta where client_idle_wait_time <> 0 or total_rqst_time <> 0 group by collection_time, member order by collection_time`

(refer to Speaker's Notes)

Loader – Sample Queries 2

- Number of active connections
  - `select collection_time, member, count(*) num_connections from ibmhist.mon_get_connection_delta where client_idle_wait_time <> 0 or total_rqst_time <> 0 group by collection_time, member order by collection_time`
- Number of active agents
  - `select collection_time, member, count(*) num_agents from ibmhist.mon_get_agent group by collection_time, member order by collection_time`

(refer to Speaker's Notes)

Loader – Sample Queries 3

- CPU usage over time
  - `select collection_time, member, cpu_user, cpu_system, cpu_idle, cpu_iowait, cpu_usage_total from ibmhist.env_get_system_resources_delta order by collection_time, member`
- Memory usage over time
  - `select collection_time, member, count(*) num_agents from ibmhist.mon_get_agent group by collection_time, member order by collection_time`

(refer to Speaker's Notes)

Quick Parser

- Located in `3_quickparser/quickparse.py`, allows to format existing data without loading to database or any other pre-processing and help viewing on a terminal screen
  - Only processes comma-delimited SQL files, not raw SYS data
- Usage
  - `cd db2histmon/3_quickparser`
  - `python3 quickparse.py -h`
  - `python3 quickparse.py -dataCollectionName MON_GET_EXTENDED_LATCH_WAIT -sourcePath ~/sqllib/db2dump/IBMHIST_SAMPLE -startDate 2021-05-04-00.00.00`

Quick Parser – Execution Example

```
$ python3 quickparse.py -dataCollectionName  MON_GET_EXTENDED_LATCH_WAIT -sourcePath
~/sqllib/db2dump/IBMHIST_SAMPLE -startDate 2021-05-04-00.00.00

COLL_TIME  MBR                                          LATCH_NAME    COUNT    TIME
"07.00.00"   0                                    "ABPDispatcher"       29       3
"07.00.00"   0                                    "ABPDistributor"      31       3
"07.00.00"   0                                       "NO_IDENTITY" 2571022 149011
"07.00.00"   0                  "SMemPool__MemLatchType__latch" 2642808 201287
"07.00.00"   0                   "SMemSet__MemLatchType__latch"  197443  22031
"07.00.00"   0                          "SQLB_BPD__bpdLatch_SX"  397565 194431
"07.00.00"   0                "SQLB_DIRTY_LIST_SET__appendLatch"       1      1
"07.00.00"   0                  "SQLB_DIRTY_LIST_SET__walkLatch"     107    121
"07.00.00"   0  "SQLB_HASH_BUCKET_GROUP_HEADER__groupLatch"    2984     34
"07.00.00"   0                   "SQLB_HATELIST_SET__hateLatch"       4      4
<…snip…>

Printing the output into a file with no wrap provides a more readable view.
```

Report Script

- Located in `4_report/report.py`, allows to generate reports like MONREPORT
  - `DBSUMMARY`, `CONNECTION`, `CURRENTAPPS`, `CURRENTSQL`, `PKGCACHE`, `LOCKWAIT`
  - Unlike `MONREPORT`, data is aggregated over collection period
- Usage
  - `cd db2histmon/4_report`
  - `pip3 install pandas` (required!)
  - `python3 report.py -h`
  - `python3 quickparse.py -dataCollectionName MON_GET_EXTENDED_LATCH_WAIT -sourcePath ~/sqllib/db2dump/IBMHIST_SAMPLE -startDate 2021-05-04-00.00.00`

Report Script – DBSUMMARY Example

```
(uncompress IBMHIST_SAMPLE_archive)
$ python3 report.py -r dbsummary -p 45 ~/sqllib/db2dump/IBMHIST_SAMPLE_archive

Collection time intervals:
  Interval   1: (2021-04-26 10:50:01) - (2021-04-26 18:20:01)
  Interval   2: (2021-04-26 18:20:01) - (2021-04-27 01:50:01)
  Interval   3: (2021-04-27 01:50:01) - (2021-04-27 09:20:00)

Work volume and throughput
--------------------------------------------------------------------------
TOTAL_APP_COMMITS          | 1:*9927.0      2: 9014.0      3:  02.0      |
<…skipping…>
Row processing
  ROWS_READ/ROWS_RETURNED  | 1: 0.84        2: 0.7         3: 0.95       |
    ROWS_READ              | 1: 343158.0    2: 307246.0    3: 389092.0   |
    ROWS_RETURNED          | 1: 408399.0    2:*405060.0    3: 409036.0   |
  ROWS_MODIFIED            | 1:*79950.0     2: 20760.0     3: 21424.0    |
```

Outlier (highest value)

Report Script – PKGCACHE Example

```
(uncompress IBMHIST_SAMPLE_archive)
$ python3 report.py -r pkgcache ~/sqllib/db2dump/IBMHIST_SAMPLE_archive

Top 10 statements by TOTAL_CPU TIME per exec
--------------------------------------------------------------------------
TOTAL_CPU_TIME PEAK TIME            STMT_TEXT
 26249.230769  2021-04-27 14:00:00   call SYSPROC.ADMIN_CMD ( ' expo...
  7314.000000  2021-04-26 22:00:00   SELECT CURRENT TIMESTAMP AS COL...
  5201.000000  2021-04-26 18:00:00   SELECT CURRENT TIMESTAMP AS COL...
  5038.000000  2021-04-26 18:00:00   SELECT CURRENT TIMESTAMP AS COL...
  4339.000000  2021-04-27 12:00:01   CALL "IBMHIST"."PROC_COLLECT"(?,?)
  4287.000000  2021-04-27 09:00:00   SELECT CURRENT TIMESTAMP AS COL...
  4248.500000  2021-04-26 11:00:01   CALL "IBMHIST"."PROC_COLLECT"(?,?)
  4161.666667  2021-04-26 18:00:00   CALL "IBMHIST"."PROC_COLLECT"(?,?)
  4154.000000  2021-04-27 05:00:00   CALL "IBMHIST"."PROC_COLLECT"(?,?)
  4139.666667  2021-04-26 12:00:00   CALL "IBMHIST"."PROC_COLLECT"(?,?)
```

Demo

- Simulate contention scenario during normal Db2 workload
- Let’s see what `db2histmon` can do for us

Demo – Setup

1. Deploy `db2histmon` on SAMPLE database:

```
$ python3 setup.py sample
```

2. Data collection starts:

```
$ ls ~/sqllib/db2dump/IBMHIST_SAMPLE
SAMPLE_2021042613

$ ls -l ~/sqllib/db2dump/IBMHIST_SAMPLE/SAMPLE_2021042613/
-rw-r--r-- 1 yunpeng pdxdb2  381 Apr 26 13:37 MON_GET_APPL_LOCKWAIT_202104261303.del
-rw-r--r-- 1 yunpeng pdxdb2  381 Apr 26 13:37 MON_GET_APPL_LOCKWAIT_202104261304.del
-rw-r--r-- 1 yunpeng pdxdb2  381 Apr 26 13:37 MON_GET_APPL_LOCKWAIT_202104261305.del
-rw-r--r-- 1 yunpeng pdxdb2  381 Apr 26 13:37 MON_GET_APPL_LOCKWAIT_202104261306.del
<…snip…>
```

Demo – Create Contention

- Inject artificial delays to get lock contention:

```
$ db2trc on -debug "DB2.SQLB.sqlbDMSMapAndRead.entry" -sleep 1
Trace is turned on
```

- MON_GET_APPL_LOCKWAIT data starts to grow:

```
$ ls -l SAMPLE_2021042613/ | grep MON_GET_APPL_LOCKWAIT
-rw-r--r-- 1 yunpeng pdxdb2  381 Apr 26 13:37 MON_GET_APPL_LOCKWAIT_202104261329.del
-rw-r--r-- 1 yunpeng pdxdb2  381 Apr 26 13:37 MON_GET_APPL_LOCKWAIT_202104261331.del
-rw-r--r-- 1 yunpeng pdxdb2  381 Apr 26 13:37 MON_GET_APPL_LOCKWAIT_202104261332.del
-rw-r--r-- 1 yunpeng pdxdb2  381 Apr 26 13:37 MON_GET_APPL_LOCKWAIT_202104261334.del
-rw-r--r-- 1 yunpeng pdxdb2 1773 Apr 26 13:37 MON_GET_APPL_LOCKWAIT_202104261335.del
-rw-r--r-- 1 yunpeng pdxdb2 1772 Apr 26 13:37 MON_GET_APPL_LOCKWAIT_202104261337.del
<…snip…>
```

Demo – Examine Locks

- Use `quickparse` to inspect what happened during the slowdown:

```
$ python3 ./3_quickparser/quickparse.py -dataCollectionName MON_GET_APPL_LOCKWAIT -sourcePath ./
-display summary -startDate 2021-04-26-13.00.00 -endDate 2021-04-26-13.45.00

  COLLECTION_TIME HLDR LOCK_WAIT_START_TIME                    LOCK_NAME    TYPE MODE RQSTD   ST
"13.36.04.720942" 7783 "13.36.04.072482" "E0AE8413987F000000000001C1" "PLAN"  "X"   "X"   "W"
"13.36.04.720942" 7783 "13.36.04.078092" "E0AE8413987F000000000001C1" "PLAN"  "X"   "X"   "W"
"13.36.04.720942" 7783 "13.36.04.080610" "E0AE8413987F000000000001C1" "PLAN"  "X"   "X"   "W"
"13.36.04.720942" 7783 "13.36.04.081075" "E0AE8413987F000000000001C1" "PLAN"  "X"   "X"   "W"
"13.36.04.720942" 7783 "13.36.04.081558" "E0AE8413987F000000000001C1" "PLAN"  "X"   "X"   "W"
"13.36.04.720942" 7783 "13.36.04.090781" "E0AE8413987F000000000001C1" "PLAN"  "X"   "X"   "W"
"13.36.04.720942" 7783 "13.36.04.096834" "E0AE8413987F000000000001C1" "PLAN"  "X"   "X"   "W"
"13.36.04.720942" 7783 "13.36.04.097443" "E0AE8413987F000000000001C1" "PLAN"  "X"   "X"   "W"
<…snip…>
```

- Lock `E0AE8413987F000000000001C1` held by handle `7783`

Demo – Examine Latches

- Determine which latches handle 7783 has owned:

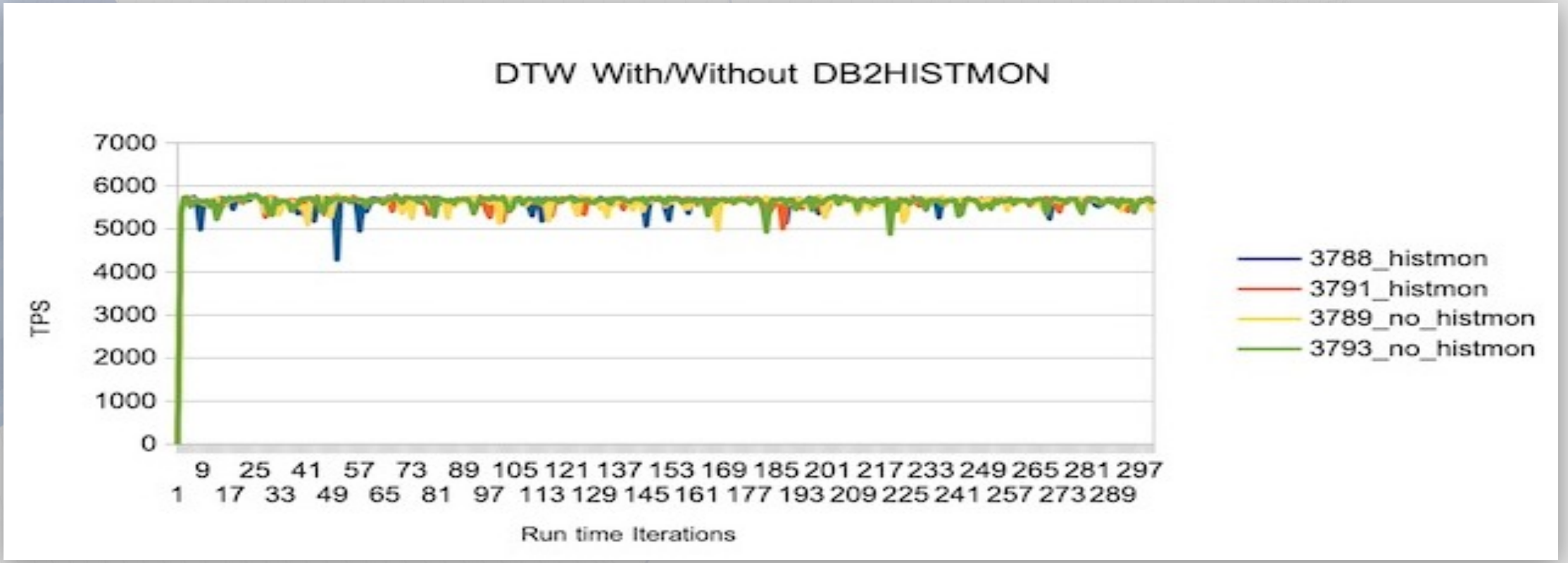
```
$ python3 ./3_quickparser/quickparse.py -dataCollectionName MON_GET_LATCH -sourcePath ./ -display
summary -startDate 2021-04-26-13.00.00 -endDate 2021-04-26-13.45.00 -applHandle 7783

     TIME                                LATCH_NAME    MEMORY_ADDRESS    EDU_ID              EDU_NAME
"13.36.04"             "SQLB_BPCB_GSS__bpcbLotch" "0x7F980E319340"     152  "db2agent (SAMPLE)"
"13.36.04" "SQLB_POOL_CB__extentAnchorTableLatch" "0x7F9811B718C0"     152  "db2agent (SAMPLE)"
"13.36.04"               "SQLB_POOL_CB__ptfLotch" "0x7F9811B5CF40"     152  "db2agent (SAMPLE)"
"13.36.04"              "SQLB_POOL_CB__readLotch" "0x7F9811B5CEC0"     152  "db2agent (SAMPLE)"
"13.36.04"        "SQLB_POOL_MAP_CB__range_latch" "0x7F9811B5CFA0"     152  "db2agent (SAMPLE)"
<…snip…>
```

- We can observe that EDU 152 associated with handle 7783 (column edited out) has frequently owned large number of table space latches over long period of time

Performance Overhead

- No observable performance degradation



Output Data Size

- Prediction: Around 200 MB – 2 GB per day, workload dependent

Level 1

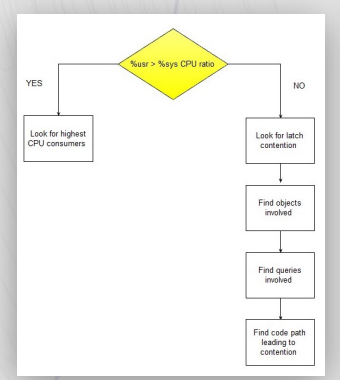
Name	Condition	Frequency (min)	KB per invocation	KB per hour	KB per day
ENV_GET_SYSTEM_RESOURCES_v11		1	0.3	20.0	481.0
ENV_CF_SYS_RESOURCES_v11	auto-detect pureScale only	1	0.0	0.0	0.0
MON_GET_AGENT_v11		2	28.8	862.8	20707.2
MON_GET_CONNECTION_v11		3	51.1	1021.9	24526.6
MON_GET_ACTIVITY_v11		2	28.8	862.7	20704.3
MON_GET_UNIT_OF_WORK_v11		2	45.0	1350.5	32411.5
MON_CURRENT_SQL_PLUS_v11		3	1.6	32.9	790.1
MON_GET_APPL_LOCKWAIT_PLUS_v11		1	0.1	6.7	159.8
MON_GET_CF_CMD_v11	auto-detect pureScale only	1	0.0	0.0	0.0
MON_GET_CF_v11	auto-detect pureScale only	1	0.0	0.0	0.0
MON_GET_CF_WAIT_TIME_v11	auto-detect pureScale only	1	0.0	0.0	0.0
MON_GET_LATCH_v11		1	0.0	0.9	21.6
MON_GET_EXTENDED_LATCH_WAIT_v11		1	0.3	19.3	462.2
MON_GET_HADR_v11	auto-detect HADR only	1	0.0	0.0	0.0
MON_GET_MEMORY_POOL_v11		1	0.7	44.4	1065.6
MON_GET_MEMORY_SET_v11		1	0.3	17.0	407.5
MON_GET_TRANSACTION_LOG_v11		1	0.1	5.6	133.9
				4244.6	101871.4

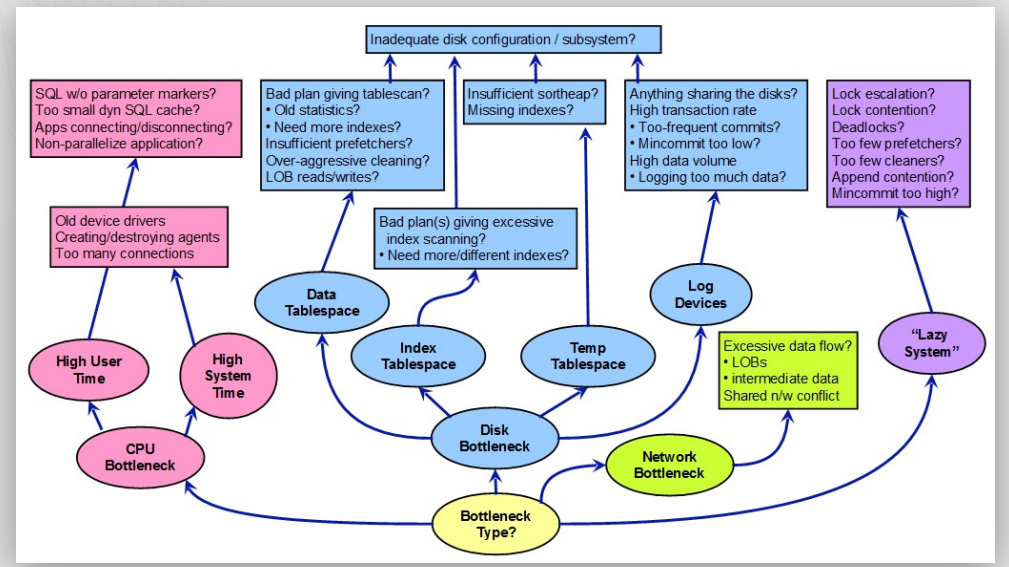
Level 2

Name	Condition	Frequency (min)	KB per invocation	KB per hour	KB per day
All of Level1, plus:				4244.6	101871.4
vmstat		1	4.0	240.0	5760.0
iostat		1	4.0	240.0	5760.0
netstat		1	44.0	2640.0	63360.0
DB_GET_CFG_v11		60	11.1	11.1	266.1
DBMCFG_v11		60	6.5	6.5	156.4
ENV_INST_INFO_v11		60	0.1	0.1	2.1
ENV_GET_REG_VARIABLES_v11		60	0.3	0.3	7.0
MON_GET_EXTENT_MOVEMENT_STATUS_v11		1	5.1	303.2	7277.8
MON_GET_GROUP_BUFFERPOOL_v11		15	0.0	0.1	3.2
MON_GET_INDEX_v11		5	51.0	612.3	14696.4
MON_GET_LATCH_v11		1	0.0	0.9	21.6
MON_GET_BUFFERPOOL_v11		15	0.8	3.2	77.2
MON_GET_PAGE_ACCESS_INFO_v11		1	0.0	0.0	0.0
MON_GET_PKG_CACHE_STMT_v11		60	131.0	131.0	3143.1
MON_GET_REBALANCE_STATUS_v11		1	0.0	0.0	0.0
MON_GET_SERVERLIST_v11		1	0.0	0.0	0.0
MON_GET_TABLE_v11		10	13.4	80.5	1933.1
MON_GET_TABLESPACE_v11		10	8.8	52.6	1262.4
MON_GET_UTILITY_v11		1	0.0	0.0	0.0
MON_GET_WORKLOAD_v11		5	0.6	7.8	186.3
				8574.3	205784.0

Future Improvements

- When data collection points finalized, revisit the infrastructure
  - ATS only runs if the database is active and healthy
  - Should this technology be turned into an instance-level daemon?
- Autonomic and machine learning logic
  - Self-detect problems in real time
  - Inform user, open a ticket?
- `db2support` integration



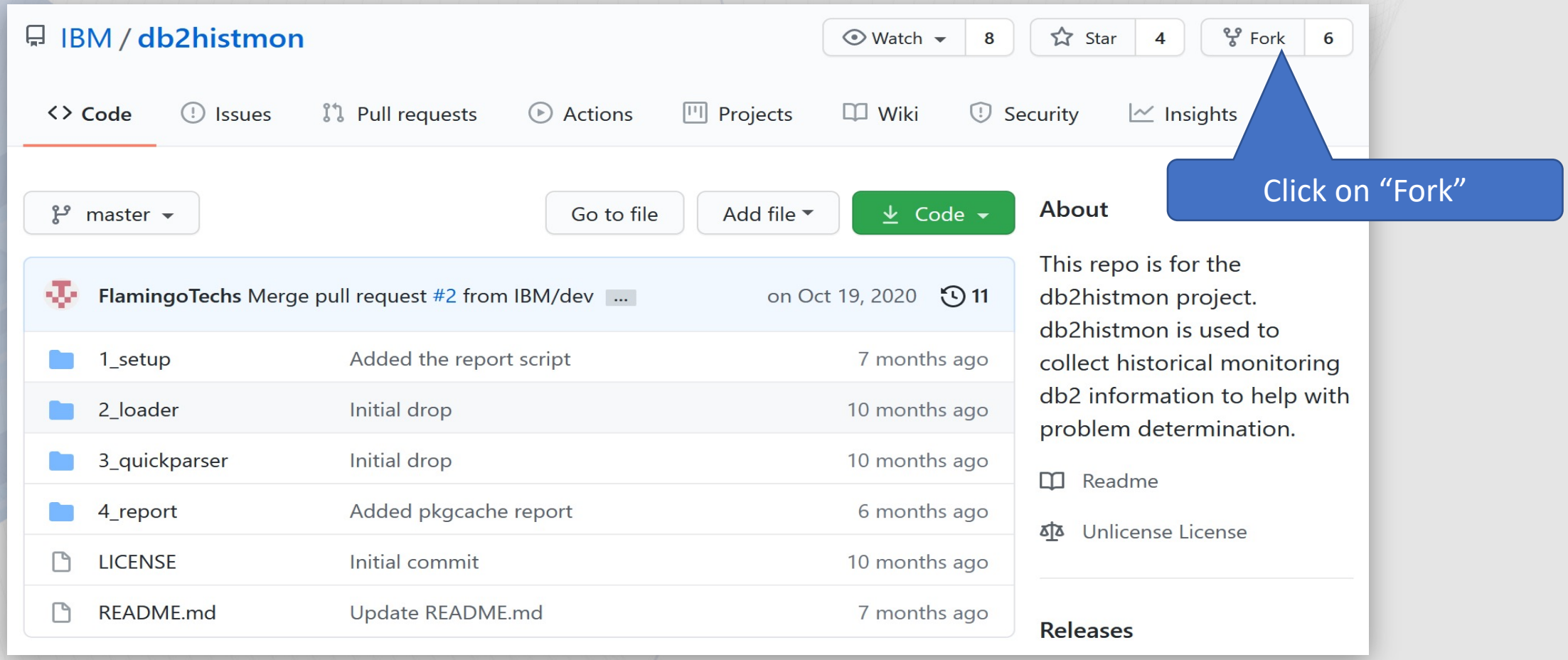


Additional Resources

- README files
  - https://github.com/IBM/db2histmon
  - https://github.com/IBM/db2histmon/tree/master/1_setup
  - https://github.com/IBM/db2histmon/tree/master/2_loader
  - https://github.com/IBM/db2histmon/tree/master/3_quickparser
  - https://github.com/IBM/db2histmon/tree/master/4_report
- David Sciarrafa's ThinkingDb2! Blog
  - http://thinkingdb2.blogspot.com/2020/06/enabling-db2-historic-monitoring.html
  - http://thinkingdb2.blogspot.com/2020/11/generating-reports-from-db2-historical.html

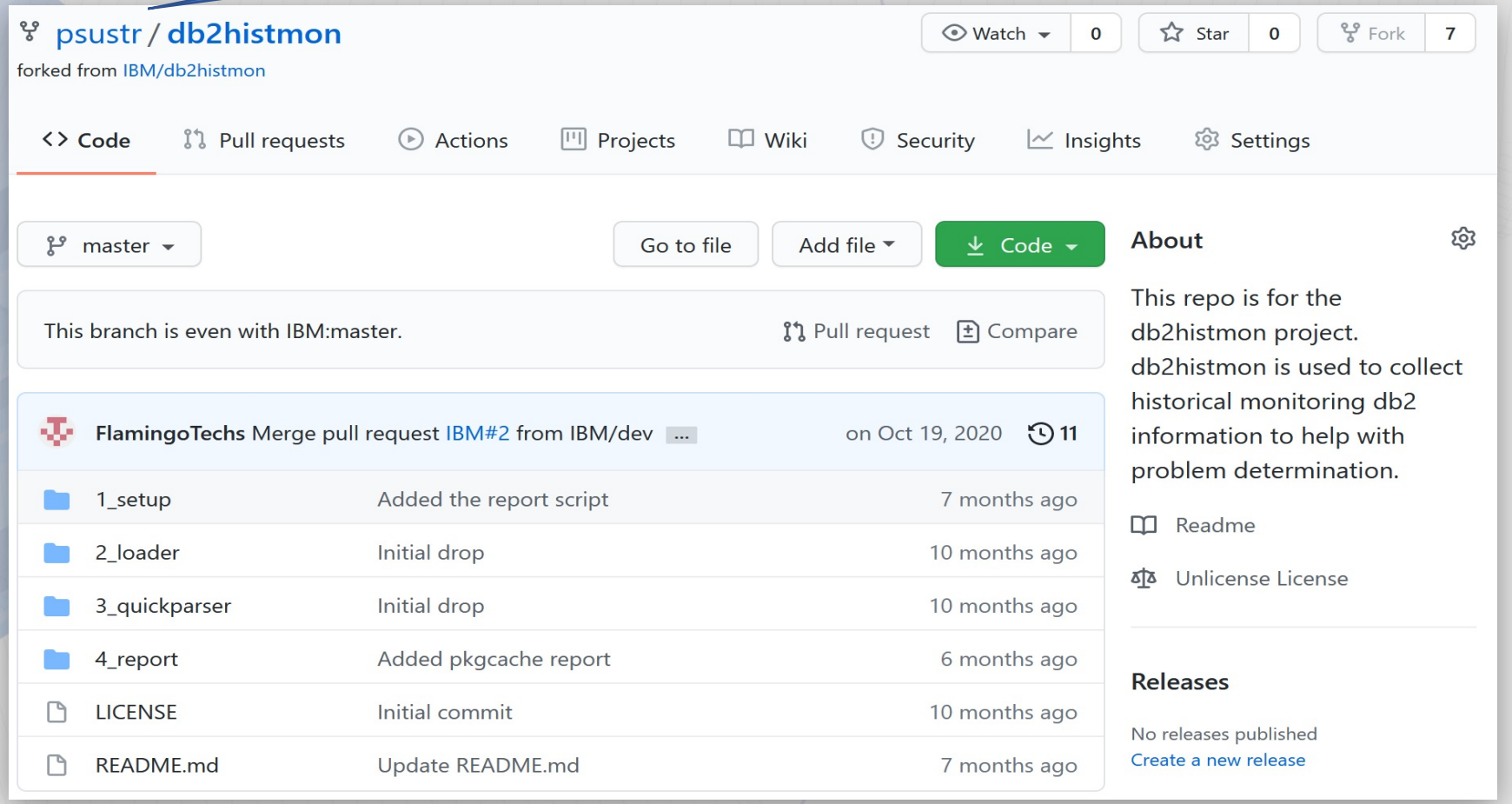
BACKUP SLIDES

Save the Best for Last: Submit Modifications!



Fork Created

Wait until the fork is created. Note that the repository name has changed from IBM/db2histmon to <yourname>/db2histmon.



Clone Repo

- Clone the newly created repo under <yourname>

```
$ git clone https://github.com/psustr/db2histmon.git

Cloning into 'db2histmon'...
remote: Enumerating objects: 60, done.
remote: Counting objects: 100% (60/60), done.
remote: Compressing objects: 100% (44/44), done.
remote: Total 60 (delta 24), reused 42 (delta 16), pack-reused 0
Receiving objects: 100% (60/60), 56.14 KiB | 746.00 KiB/s, done.
Resolving deltas: 100% (24/24), done.
```

- At this point you should see a new `db2histmon` directory
- You are welcome to make local modifications

Make Changes

- Make local changes, for example by editing `LICENSE` ☺

```
$ cd db2histmon

$ vi LICENSE

>>> Free the world!!! <<<

This is free and unencumbered software released into the public domain.

Anyone is free to copy, modify, publish, use, compile, sell, or
distribute this software, either in source code form or as a compiled
binary, for any purpose, commercial or non-commercial, and by any

"LICENSE" 26L, 1230C written
```

Commit Changes

```
$ git status -uno
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   LICENSE

no changes added to commit (use "git add" and/or "git commit -a")

$ git add -u
(hostname) /home/psustr/db2histmon

$ git commit -m "Updated the license"
[master b02f5f3] Updated the license
 1 file changed, 2 insertions(+)
```

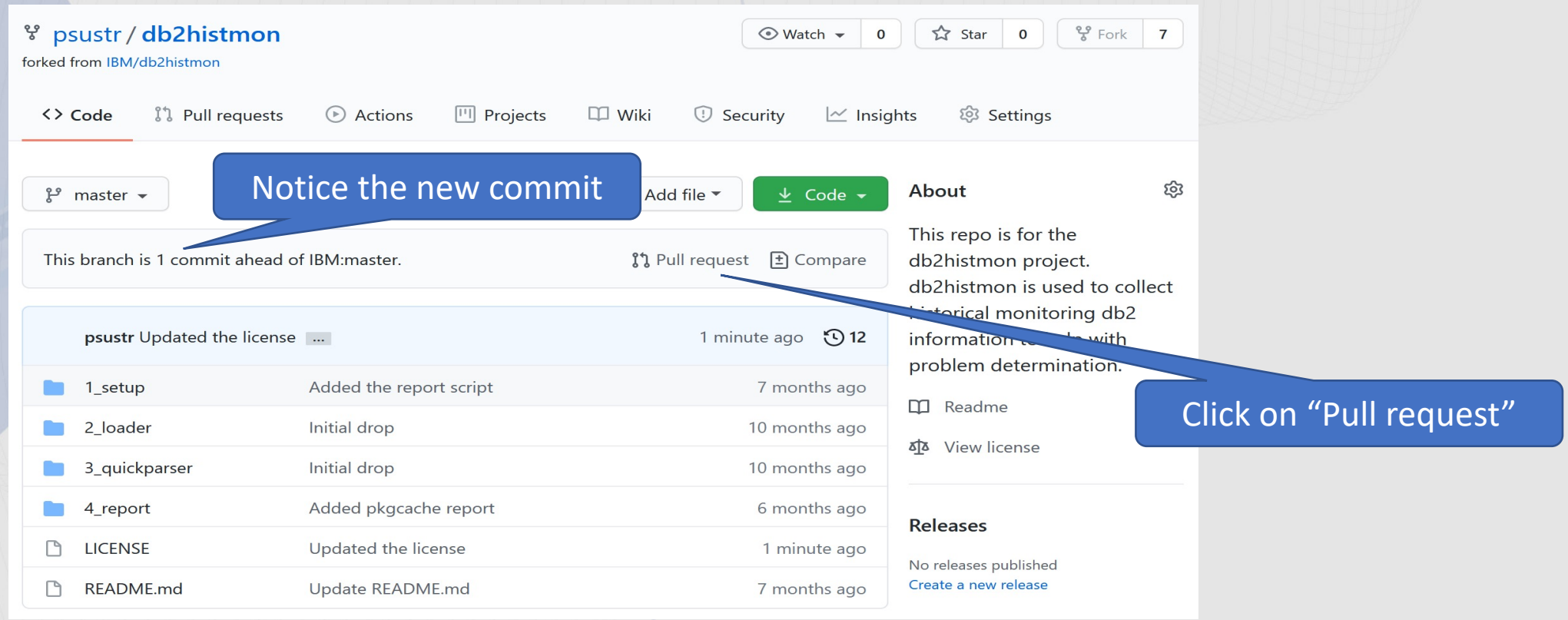
Upload Changes

- Upload the local changes to your repo at www.github.com
  - Note that username/password will be insufficient soon, 2FA will be needed

```
$ git push origin master
Username for 'https://github.com': psustr
Password for 'https://psustr@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 302 bytes | 302.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/psustr/db2histmon.git
   8bedadc..b02f5f3  master -> master
```

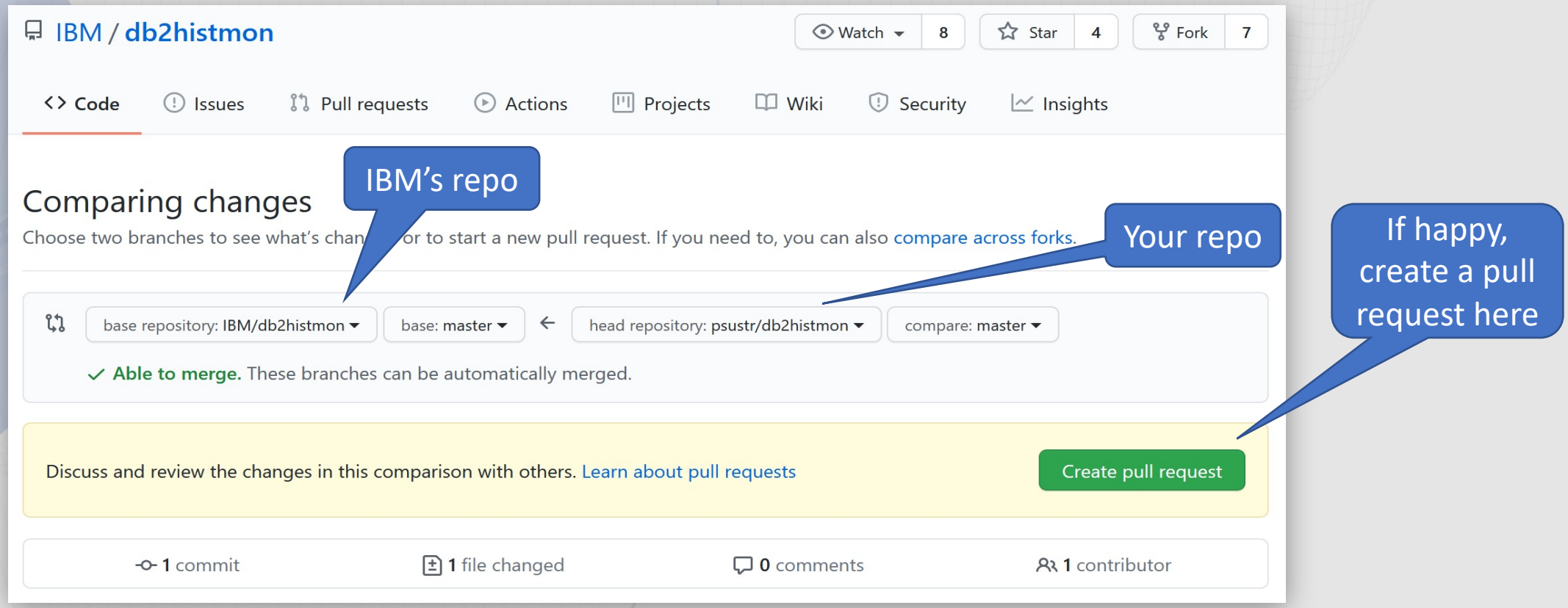
Create PR

- Create a pull request to notify IBM of your proposed changes



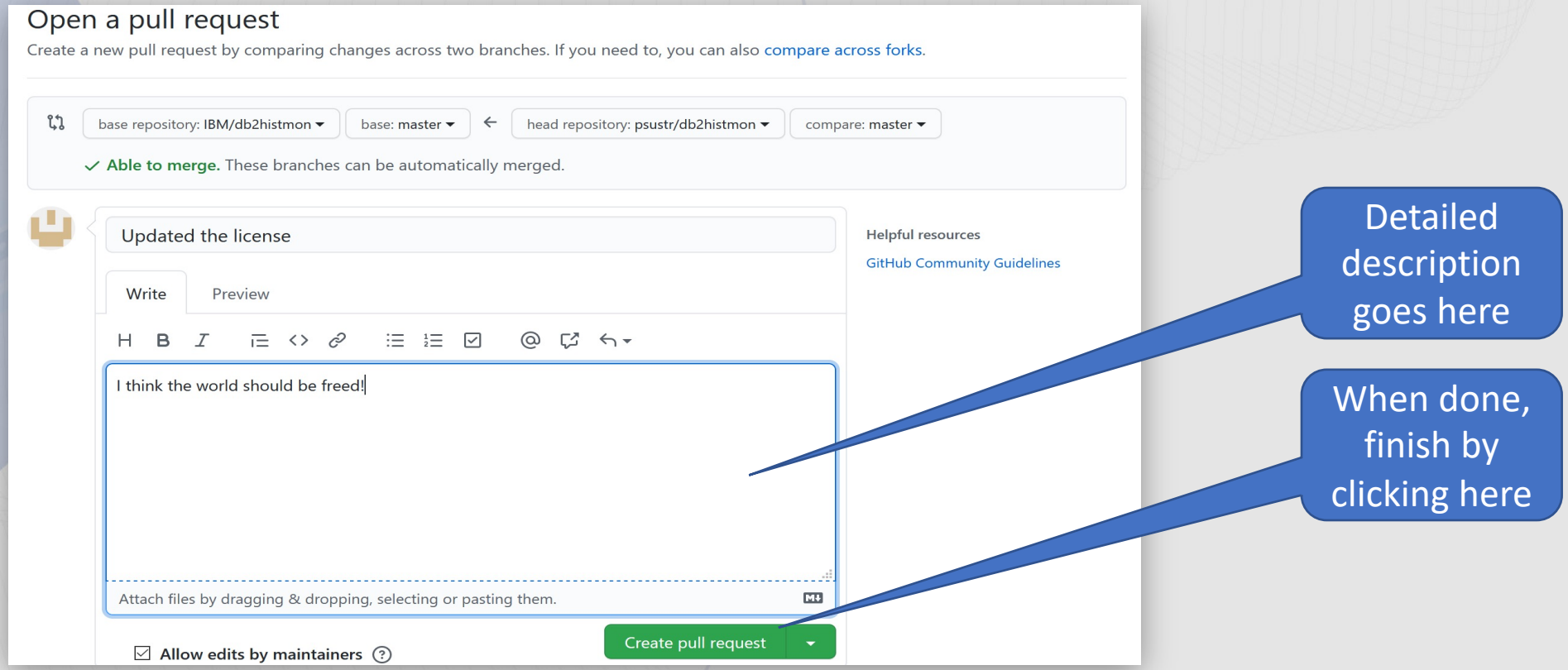
Check Merge Options

- Make sure you are asking to get your repo pulled into IBM's one



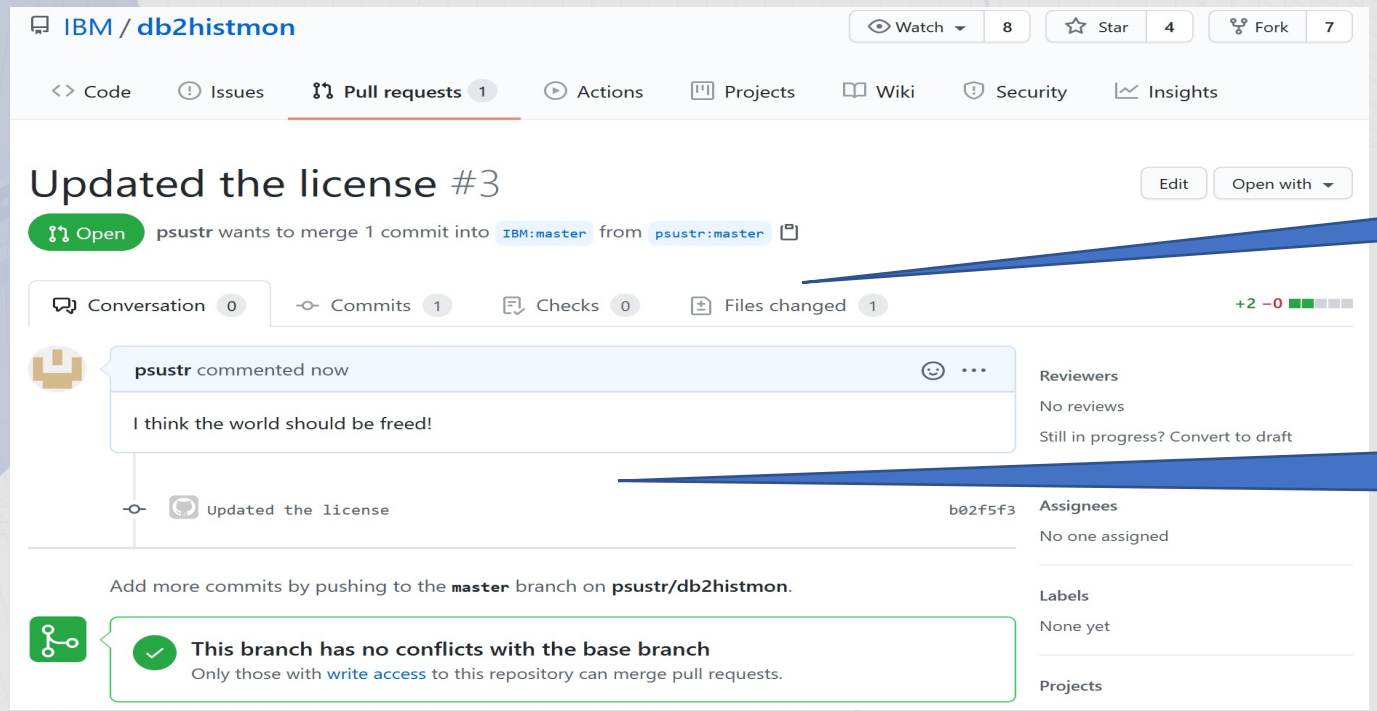
Describe Changes

- Document change reasons, and what testing has been done



Congratulations, done!

- Visit https://github.com/IBM/db2histmon/pulls to see your PR
- The next step is up to IBM to either approve or reject the change

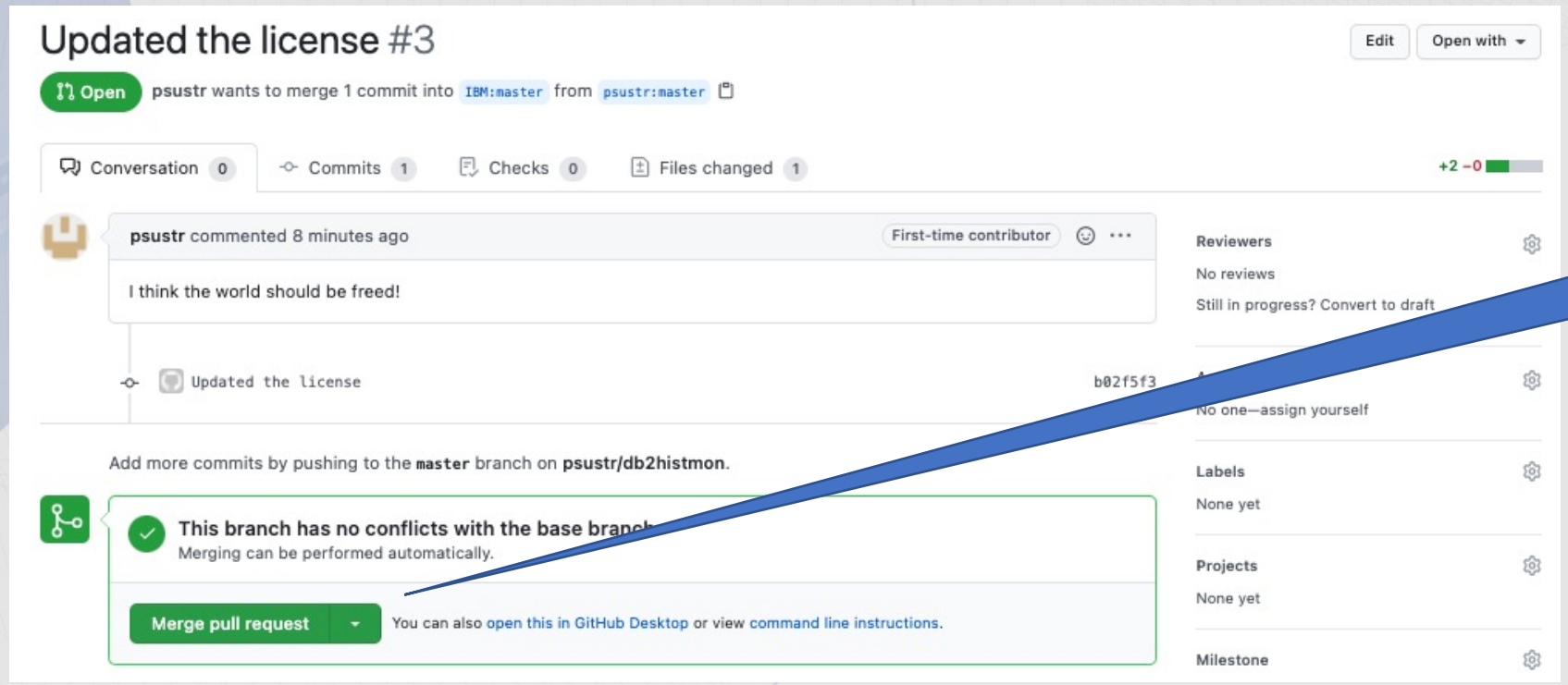


Find your changes here

Conversation between you and IBM; IBM may ask for more information

IBM Approves

- IBM reviews and approves the change, if appropriate
- Congratulations, the change is now official!



Merge will include the change to the official repo

Speakers: Pavel Sustr, Michael Wang
Company: IBM Canada Lab
Emails: psustr@ca.ibm.com, yunpeng@ca.ibm.com
@pavel_sustr

Session code: 7204

Please fill out your session evaluation before leaving!!!!