Journey from RDS -> Aurora -> Community PostgreSQL

Arcesium

- A unified data platform
- Powerful Domain-Aware Capabilities
- Transformative Business Outcomes

Agenda

- PostgreSQL evaluation
- Managed hosting vs self-hosting
- RDS PostgreSQL -> Aurora PostgreSQL
- Managing Aurora PostgreSQL
- Aurora Cost optimizations
- Community PostgreSQL

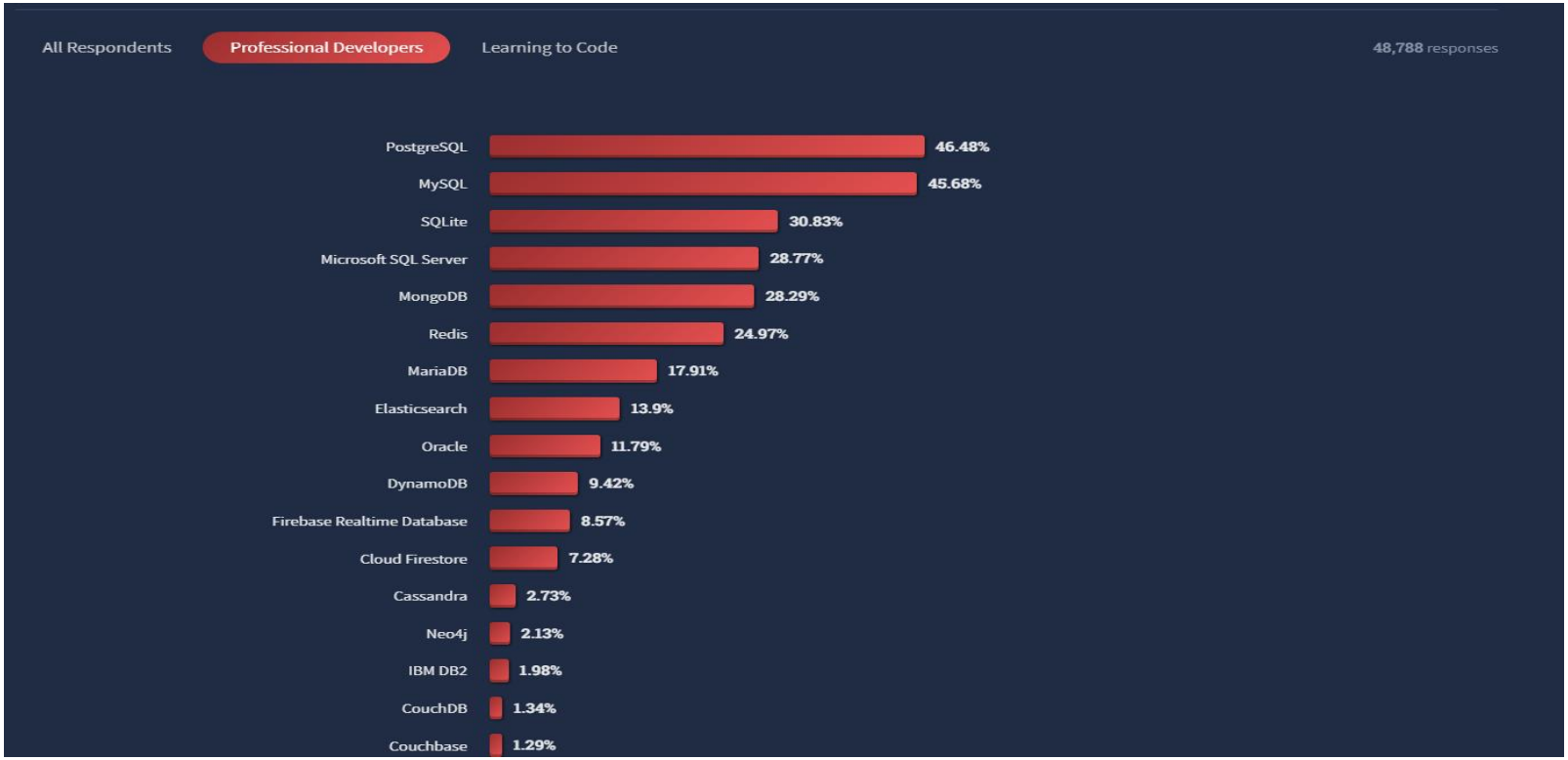
PostgreSQL evaluation

Why PostgreSQL?

- World's most advanced open-source relational database
- DBMS of the year for 2017, 2018 and 2020 (DB-engines.com)
- Highly scalable, secure, reliable, robust and cost effective
- More permissible open-source license
- Hosted by all major cloud providers (AWS, Google Cloud, Azure)
- Constantly evolving with regular releases
- Great community engagement

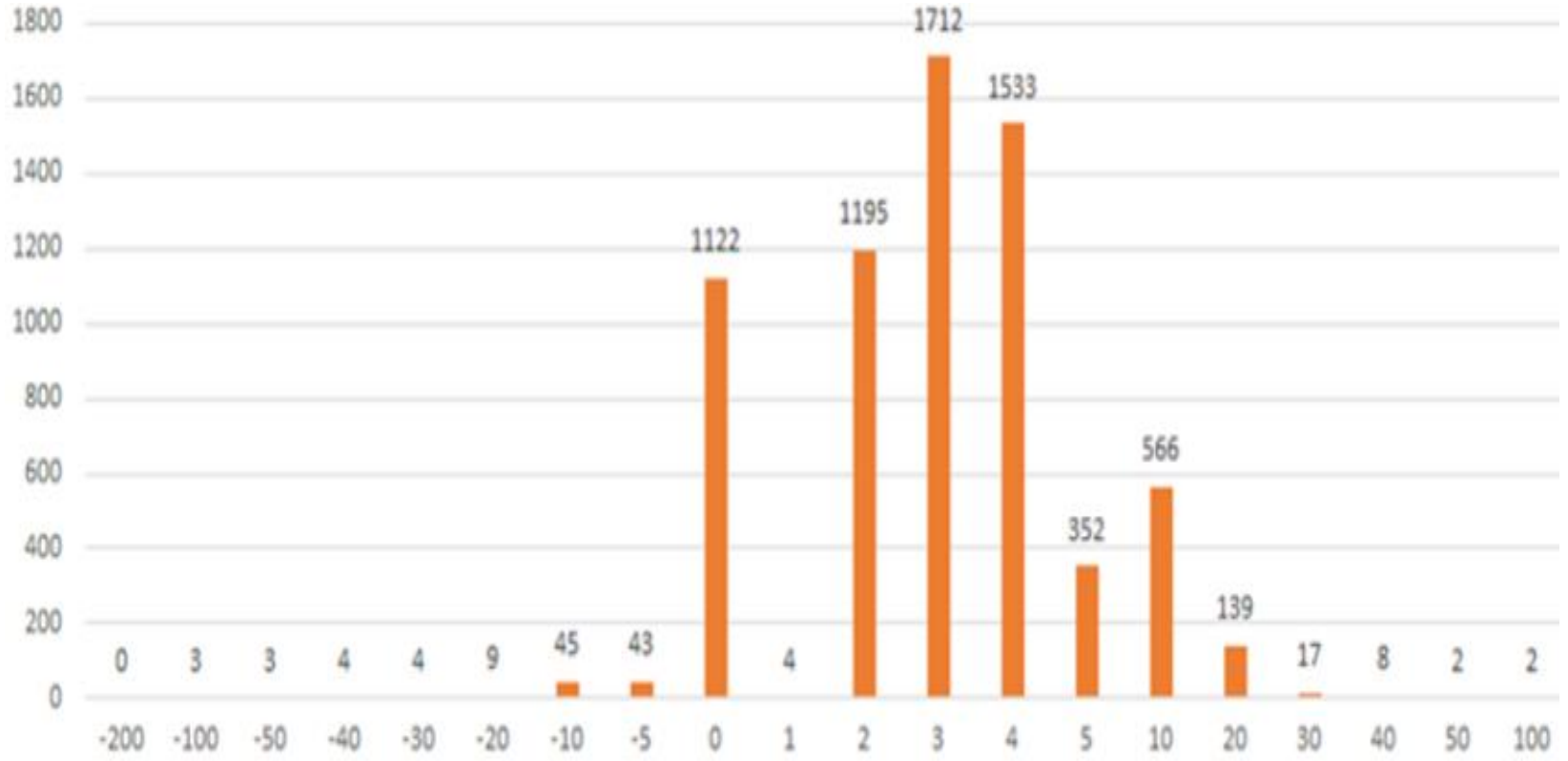
PostgreSQL evaluation

- Most preferred database by professional developers according to 2022 stackoverflow developer survey



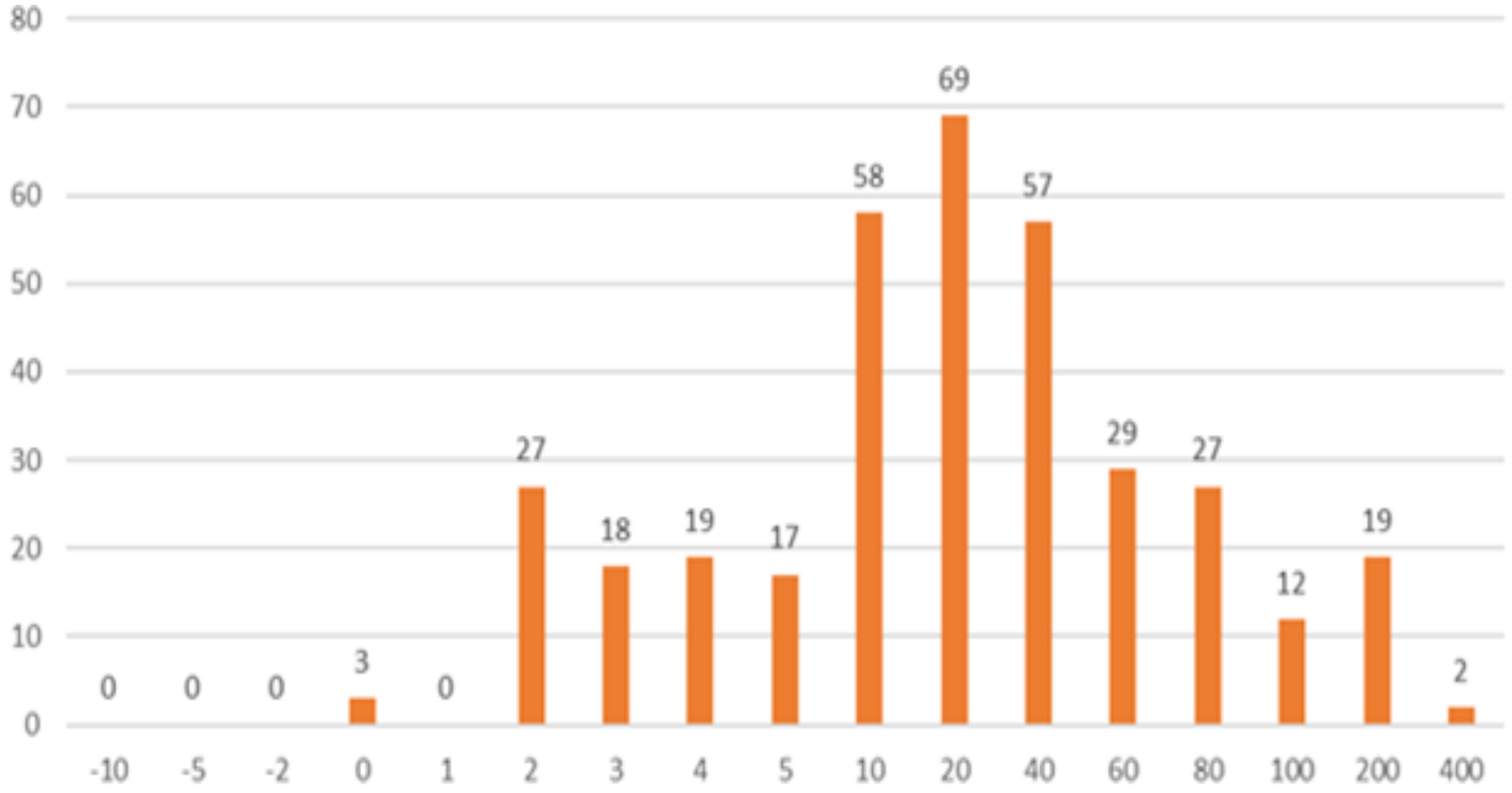
PostgreSQL evaluation

Reads on SQL Server vs PostgreSQL



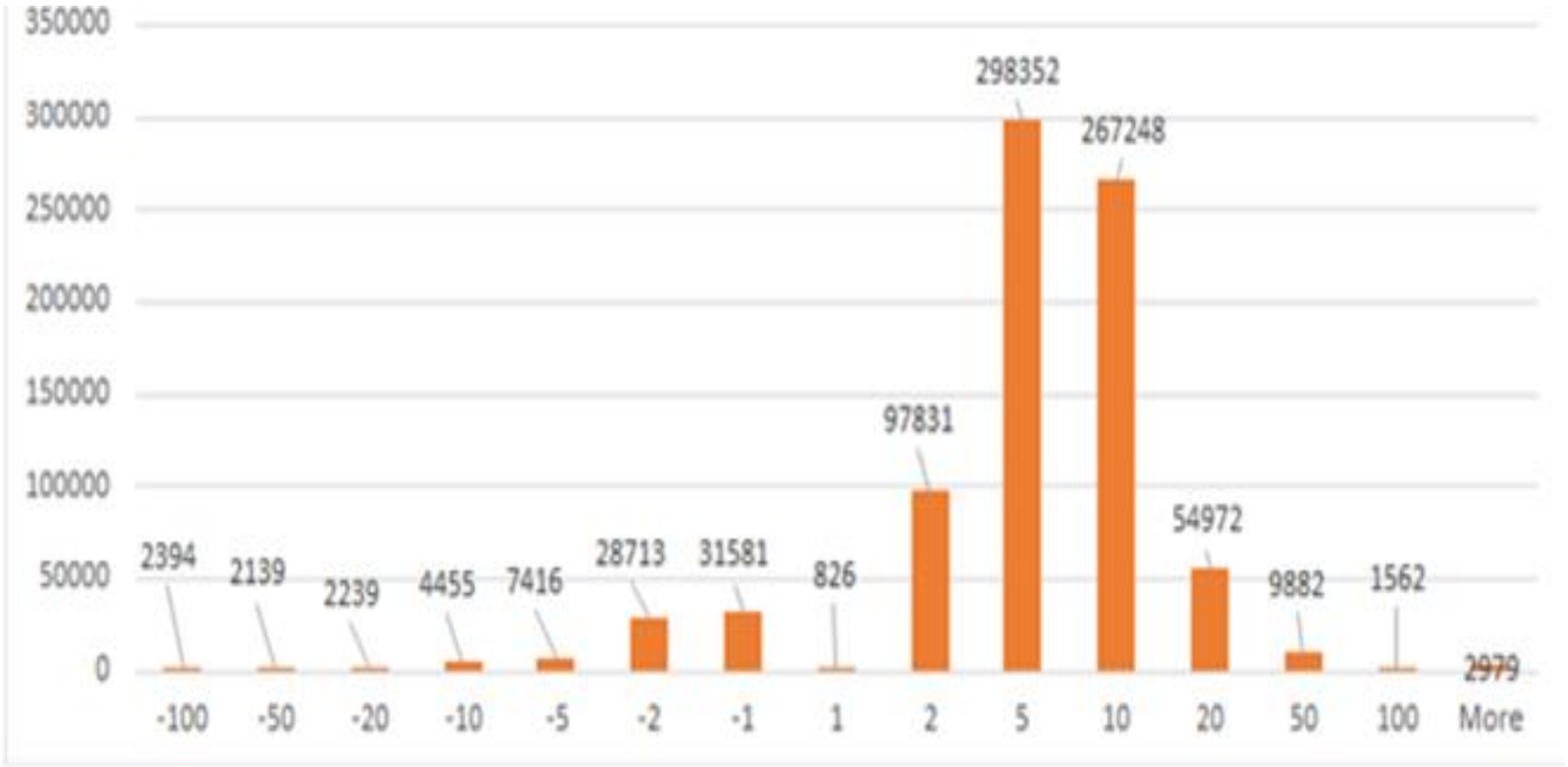
PostgreSQL evaluation

Writes on SQL Server vs PostgreSQL



PostgreSQL evaluation

Mixed workloads on SQL Server vs PostgreSQL



Why RDS

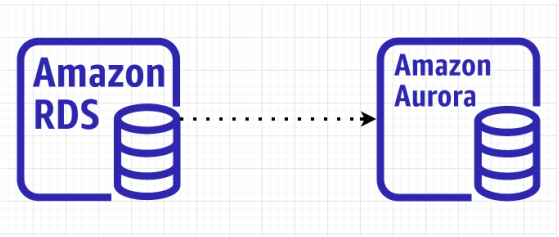
- Ease of creation and managing the instances
- Ease of scaling
- Ease of configuring HA, backups etc.
- Quickly getting off the ground

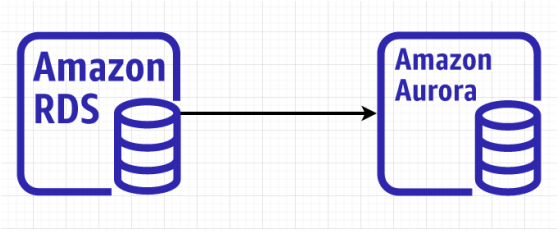
Why we moved from RDS to Aurora

- Key features of Aurora PostgreSQL
  - High availability across 3-AZ
  - Auto recovery of failed node
  - Copy on write clones
  - Auto scaling of storage and IOPS
  - Reader endpoint to load balance
  - Incremental backups

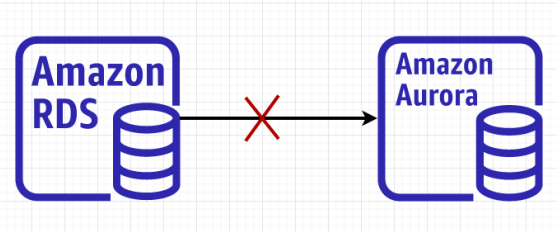
How we moved from RDS to Aurora

- Create Aurora read replica for the existing RDS instance of 9.6.6 version.
- Wait for synchronization
- Promote Aurora as primary





- Observations post cutover to Aurora:
  - Access to GIN, GiST indexes made system unstable
  - Increase in CPU usage



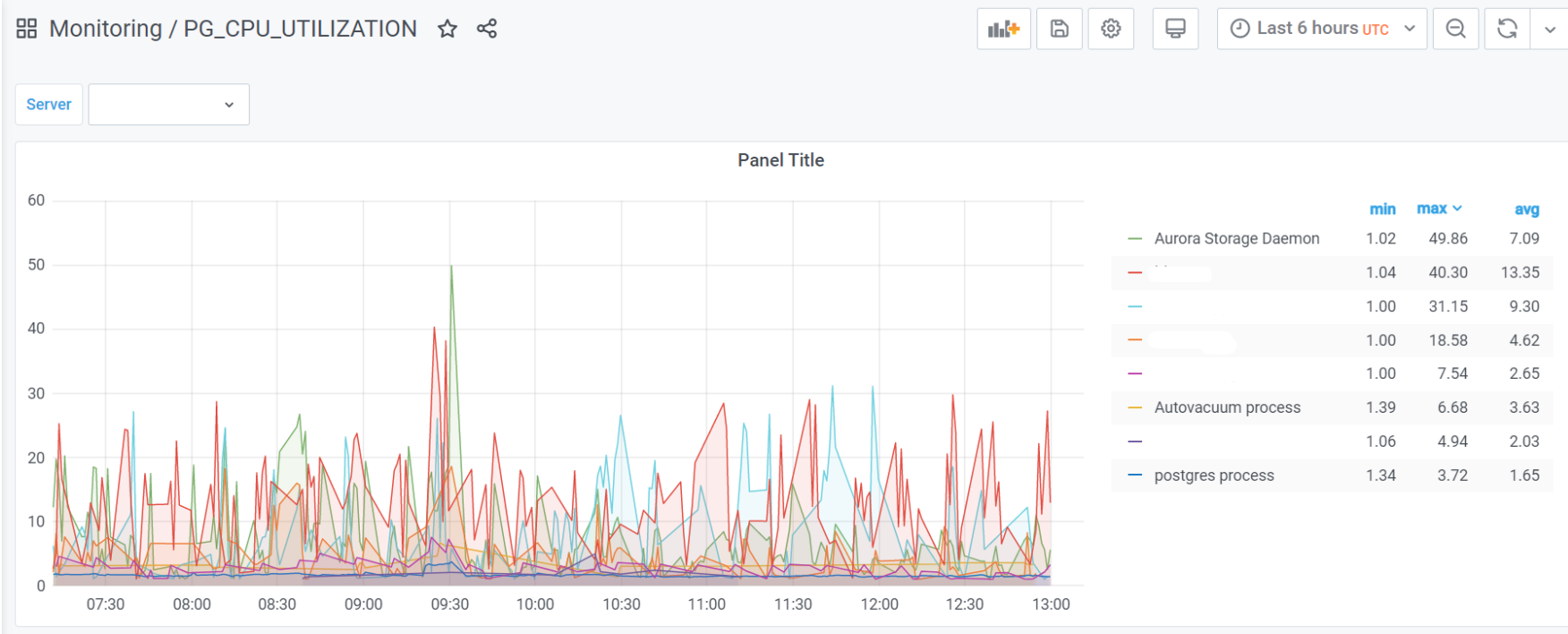
Managing Aurora

How we built automation to significantly improve developer experience while saving operational time

- Access to AWS Performance Insights
- Enable query logging and extensions - auto_explain, pg_stat_statements

Managing Aurora

- Enable enhanced monitoring to capture the os metrics to cloudwatch.
- User level CPU usage to find which user consuming more CPU in an Aurora instance



Managing Aurora

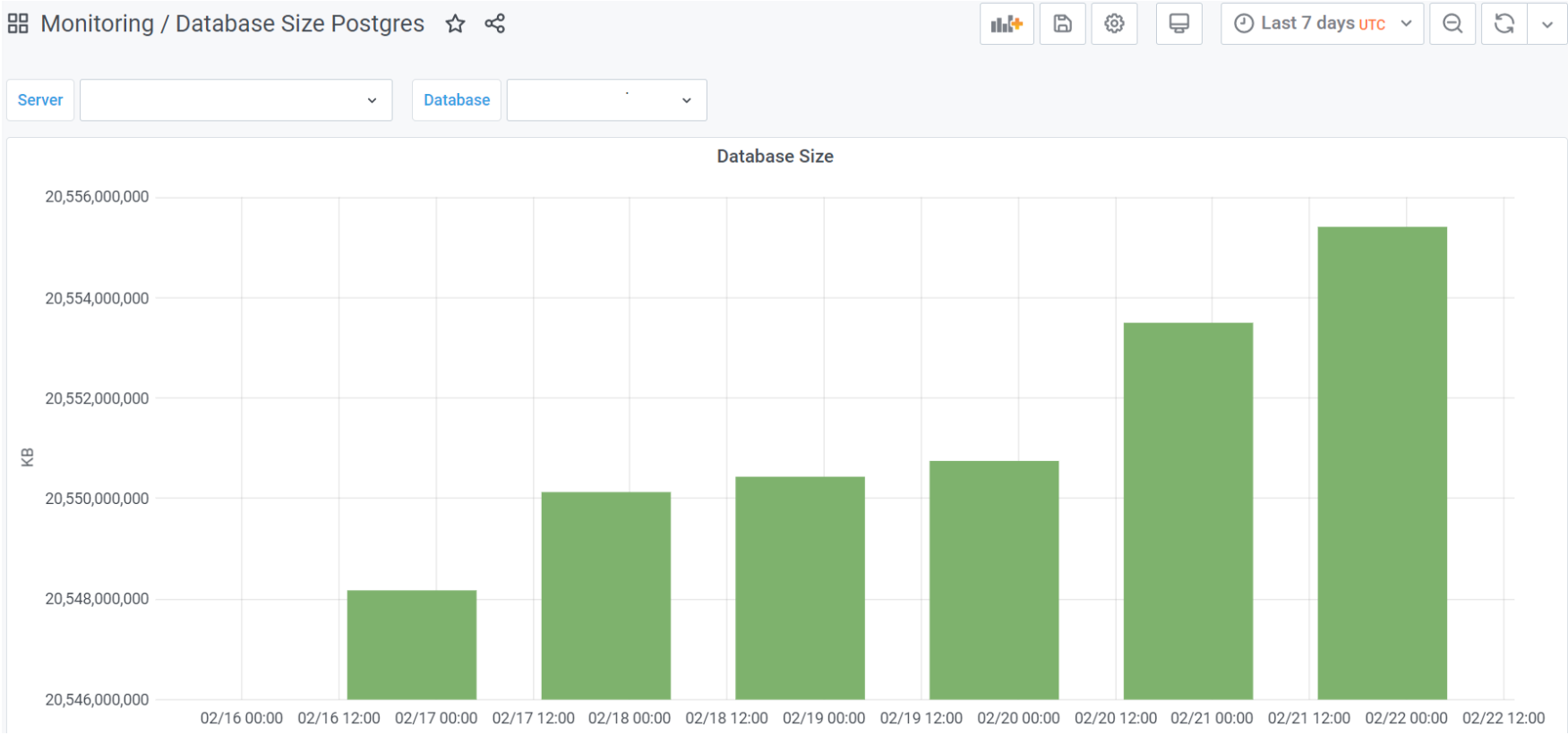
- Enable pg_stat_activity_history
- Monitoring dashboard with query level resource usage

Process Details

date_time_et	server_name	database_name	user_name	process_type	process_name	pid	average_cpu_used_pct	average_memory_used_pct	query
2023-02-22T08:05:00Z				postgres:	postgres idle	798	6.15	1.73	INSERT INTO extraction_status(\n tenant_id\,\n identity\,\n created_utc\,\n updated_utc\,\n raw_
2023-02-22T07:37:00Z				postgres:	postgres idle	18802	4.18	1.46	SELECT\n etl_file_alias\n \,etl_dataset_type_id\n \,etl_parser_id\n \,cocoa_parser_id\n \,in_cocoa
2023-02-22T07:37:00Z				postgres:	postgres idle	19087	4.05	0.41	SELECT typinput='array_in'::regproc as is_array\, typtype\, typname FROM pg_catalog.pg_type LEF

Managing Aurora

- Database size growth trend



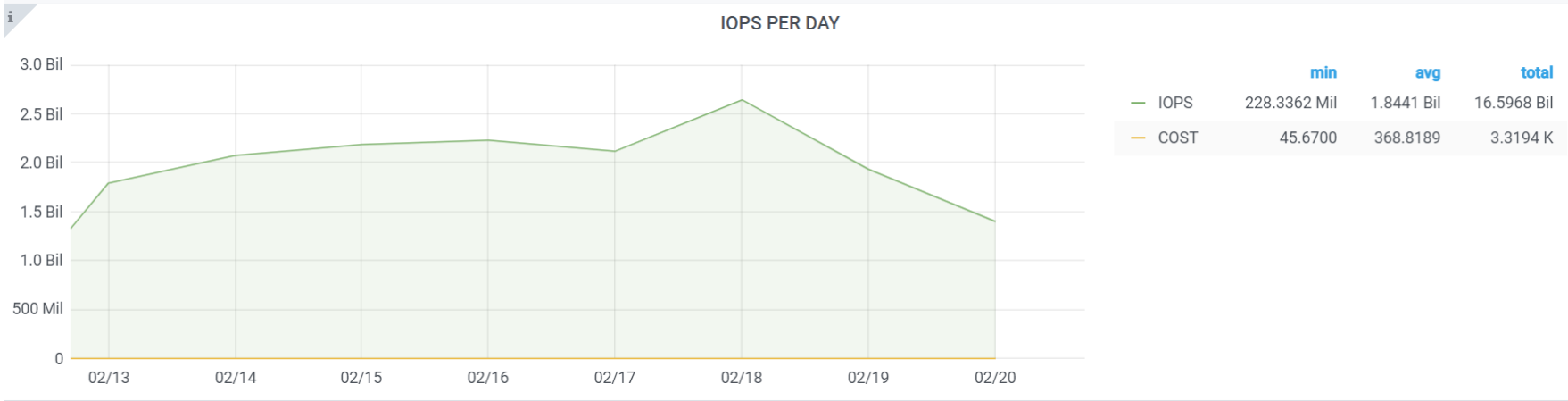
Managing Aurora

- Dashboard with IOPS usage of Aurora cluster

IOPS Per Instance

DATE	instancename	IOPS	COST
2023-02-16T00:00:00Z		624.36 Mil	124.87
2023-02-17T00:00:00Z		567.09 Mil	113.42
2023-02-15T00:00:00Z		559.66 Mil	111.93
2023-02-19T00:00:00Z		526.15 Mil	105.23
2023-02-18T00:00:00Z		522.40 Mil	104.48

1 2 3 4 5 6 7 8 9



Managing Aurora contd..

- Aurora clones to estimate time of database upgrades
- Aurora clones along with pgreplay to assess upgrade version performance
- Automations to upgrade or patch database fleet with minimal manual intervention

Managing Aurora Costs

- Make best use of AWS Cost Explorer
- Look out for cost saving features
  - Storage costs: copy-on-write clones
  - Compute costs: Instance reservations & Graviton processors
  - IOPS costs
- Build infrastructure around database workloads
  - Collect historical data of pg_stat_activity
  - Look for Wait Events incurring IOPS costs
    - IO:DataFileRead
    - LWLock:buffer_mapping
  - Actions:
    - Indexing
    - Partitioning
    - Instance resizing

Managing Aurora Costs contd...

- pgreplay for performance benchmarking with prod similar workload
- R5 vs R6g observations:
  - CPU is better performant in R6g
  - Cost is 10.5% cheaper
  - Test and validate performance for your workloads
- Migrated our Aurora instances to R6g which saved the cost and improved the performance

Preparing for self-managed

- Built the team and have more control on the environment
- Migration:
  - Logical Replication
  - pglogical 2
- High Availability Infrastructure:
  - Streaming Replication + Repmgr + keepalived
  - Streaming Replication + Patroni + etcd + HAProxy
- Backup:
  - pgBackRest
- Monitoring:
  - pgwatch2, extended as per custom needs

Why are we moving to self-managed

- As data grows, and usage increases, IOPS costs will increase significantly
- Limited local storage on Aurora limiting maintenance operations
- Interesting projects:
  - https://github.com/citusdata/citus
  - https://github.com/hydradatabase/hydra
  - https://github.com/StarfishStorage/explain-running-query

Questions?

Thank you!