

001

002

003

004

005

006

007

008

009

010

011

012

013

014

015

016

017

018

019

020

021

022

023

024

025

026

027

028

029

030

031

032

033

034

035

036

037

038

039

040

041

042

043

044

045

046

047

048

049

050

051

052

053

054

055

056

057

058

059

060

061

062

063

064

065

066

067

068

069

070

071

072

073

074

075

076

077

078

079

080

081

082

083

084

085

086

087

088

089

090

091

092

093

094

095

096

097

098

099

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

1001

1002

1003

1004

1005

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

1018

1019

1020

1021

1022

1023

1024

1025

1026

1027

1028

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

1134

1135

1136

1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

1188

1189

1190

1191

1192

1193

1194

1195

1196

1197

1198

1199

1200

1201

1202

1203

1204

1205

1206

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1217

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1251

1252

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

1266

1267

1268

1269

1270

1271

1272

1273

1274

1275

1276

1277

1278

1279

1280

1281

1282

1283

1284

1285

1286

1287

1288

1289

1290

1291

1292

1293

1294

1295

1296

1297

1298

1299

1300

1301

1302

1303

1304

1305

1306

1307

1308

1309

1310

1311

1312

1313

1314

1315

1316

1317

1318

1319

1320

1321

1322

1323

1324

1325

1326

1327

1328

1329

1330

1331

1332

1333

1334

1335

1336

1337

1338

1339

1340

1341

1342

1343

1344

1345

1346

1347

1348

1349

1350

1351

1352

1353

1354

1355

1356

1357

1358

1359

1360

1361

1362

1363

1364

1365

1366

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

1404

1405

1406

1407

1408

1409

1410

1411

1412

1413

1414

1415

1416

1417

1418

1419

1420

1421

1422

1423

1424

1425

1426

1427

1428

1429

1430

1431

1432

1433

1434

1435

1436

1437

1438

1439

1440

1441

1442

1443

1444

1445

1446

1447

1448

1449

1450

1451

1452

1453

1454

1455

1456

1457

1458

1459

1460

1461

1462

1463

1464

1465

1466

1467

1468

1469

1470

1471

1472

1473

1474

1475

1476

1477

1478

1479

1480

1481

1482

1483

1484

<

the health and performance of databases. By querying predefined statistics views, users can gain visibility into various aspects of their databases, including read and write query throughput, performance, replication and reliability, and resource utilization. Some of the key statistics views available in PostgreSQL are:

1. **pg_stat_database:** This view collects statistics about each database in the cluster, including the number of connections, commits, rollbacks, and rows/tuples fetched and returned.
2. **pg_stat_user_tables:** This view offers statistics on user-defined tables, such as the number of rows inserted, updated, or deleted, as well as information on table size and cache usage.
3. **pg_stat_user_indexes:** This view provides statistics on user-defined indexes, including metrics related to index scans, index size, and cache usage.
4. **pg_stat_bgwriter:** This view offers statistics on the background writer process, which is responsible for managing the write-back process to disk. It provides information on buffer allocations, writes, and evictions.
5. **pg_statio_user_tables:** This view displays a cumulative count of blocks read, the number of blocks that were hit in the shared buffer cache, as well as other information about the types of blocks that were read from each table.

By leveraging these predefined statistics views, users can gain insights into the performance and resource utilization of their PostgreSQL databases, allowing them to optimize their database configuration and performance tuning for better overall database health.

2.2 QueryPilotX's Statistics

Following are the statistics provided by us.

2.2.1 Database Specific Metrics

1. **get_stats_disk_usage_for_database:** This metric provides insights into the size of each database in bytes, helping us understand the storage footprint of the databases in a clear and concise manner.
2. **get_stats_tx_rate_for_database:** This metric offers a comprehensive view of the rate of

transactions executed per second, as well as the rate of rollbacks executed separately since the last function call. This information helps us monitor the efficiency and effectiveness of the database operations with ease.

3. get_stats_seconds_since_last_vacuum_per_table:

This metric presents the amount of time passed in seconds since the last vacuum was performed on each table in the database, providing us with valuable information on when tables were last cleaned up and optimized for performance.

4. get_stats_oldest_transaction_timestamp:

This metric highlights the duration for which the longest-running transaction has been opened in the database. While this value should ideally be close to zero, an increase may indicate potential issues, such as unclosed maintenance connections, that require prompt attention to ensure optimal database performance.

5. get_stats_index_hit_rates:

This metric offers insights into the usage of indexes versus sequential scan through the table, helping us understand the effectiveness of our indexing strategies. With a clear understanding of the data and index usage, we can make informed decisions on when high or low index usage is desirable for optimal database performance.

6. get_stats_table_bloat:

This metric reveals the amount of wasted space in the database table due to the MVCC process. By identifying obsolete data that is marked as free but not truly deleted, this metric helps us identify and address table bloat issues, and optimize the database storage. The current implementation of this metric may be resource-intensive and can be disabled if necessary to avoid performance issues.

2.2.2 Database Cluster (Global) Metrics

1. get_stats_client_connections:

This insightful metric provides visibility into the current number of connections open to the database, including the actual metrics for gathering connections. It's worth noting that having an excessive number of open connections, typically over a hundred, can be detrimental.

171	2. get_stats_lock_statistics: This crucial met-	the get_multixact_members_usage_ratio).	219
172	ric sheds light on the locks that queries are	This metric can be useful in identifying po-	220
173	waiting on, as well as the locks that have been	tential multixact ID exhaustion issues and op-	221
174	granted. Extended wait times for locks can	timizing multixact usage in the PostgreSQL	222
175	indicate potential issues like heavy lock con-	tention, and warrant further investigation.	223
176			
177	3. get_stats_heap_hit_statistics: These valu-	2.2.3 Real time query analysis	224
178	able metric offers insights into the reads hit-	<u>querypilotX</u> presents an intricate and comprehen-	225
179	ting the memory buffers and disk (or disk	sive overview of real-time query monitoring met-	226
180	caches) in our cluster. Additionally, it calcu-	rics for PostgreSQL, empowering database admin-	227
181	lates the heap hit ratio based on these values.	istrators and developers with valuable insights into	228
182	It's important to note that the read amounts	query performance and resource utilization.	229
183	refer to blocks read, not actual read queries.		
184	Comparing these values with the transaction	1. One of the key metrics monitored by	230
185	rate can provide a solid understanding of the	<u>querypilotX</u> is the query duration, which re-	231
186	reading activity on the database.	veals the time duration for which a query has	232
187		been running. This enables users to swiftly	233
188	4. get_stats_replication_delays: This critical	identify queries that may be running longer	234
189	metric provides information on the replication	than expected, potentially impacting perfor-	235
190	delay in bytes between the master and each	mance.	236
191	slave. In the synchronous replication state, the		
192	replication delay is zero. Monitoring this met-	2. Furthermore, <u>querypilotX</u> provides in-depth	237
193	ric can help identify any potential replication	metrics for CPU usage, memory usage, disk	238
194	lag issues that may impact data consistency	I/O, and network activity for each active query.	239
195	and integrity.	These metrics offer visibility into resource	240
196		utilization, enabling users to pinpoint perfor-	241
197	5. get_stats_wal_file_amount: This metric of-	mance bottlenecks and identify queries that	242
198	fers visibility into the number of files in the	may be consuming excessive resources.	243
199	database cluster's Write-Ahead Log (WAL)		
200	log directory (pg_wal or pg_xlog). A sudden	3. CPU usage is represented as a percentage of	244
201	increase in the WAL file amount may indicate	CPU time utilized by each active query, help-	245
202	issues with the WAL archiving process, which	ing users identify queries that may be dispro-	246
203	could result in disk filling up and ultimately	portionately using CPU resources, thereby af-	247
204	crashing the database cluster.	fecting performance.	248
205			
206	6. get_xid_remaining_ratio,	4. Memory usage is depicted as the amount of	249
207	get_multixact_remaining_ratio,	memory being utilized by each active query,	250
208	get_multixact_members_remaining_ratio:	along with the percentage of total memory	251
209	These insightful metrics provide the remain-	being used by PostgreSQL processes overall.	252
210	ing percentage of transaction IDs ("xid"),	This aids in identifying queries that may be	253
211	multixact IDs ("mxid"), and multixact	consuming excessive memory resources, po-	254
212	members available for use by PostgreSQL	tentially causing performance issues or mem-	255
213	before exhaustion. Monitoring these metrics	ory leaks.	256
214	can help ensure that the vacuuming process is		
215	functioning as intended, maintaining optimal	5. <u>querypilotX</u> also provides insights into disk	257
216	performance for the PostgreSQL instance.	I/O and network activity, showcasing the num-	258
217		ber of bytes being read and written by each	259
218	7. get_multixact_members_per_mxid: This	active query, along with the rate of data trans-	260
	metric provides information on the num-	fer. This assists in identifying queries that	261
	ber of multixact members per multixact ID.	may be causing excessive disk I/O or network	262
	A higher number may indicate a faster de-	traffic, potentially impacting performance.	263
	pletion of multixact members (as seen in		
		6. In summary, <u>querypilotX</u> 's real-time query	264
		monitoring metrics offer a robust and in-	265
		valuable tool for monitoring PostgreSQL	266

267
268
269

270

271
272
273
274
275
276
277
278
279
280

281
282

283
284
285
286

287
288
289
290
291
292
293
294
295

296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313

databases, enabling users to proactively identify and resolve performance issues in real-time.

2.3 Analysis

After collecting the metrics related to Read query throughput and performance, write query throughput and performance, replication and reliability, and resource utilization from PostgreSQL’s statistics collector and other native sources, the next step is to analyze these metrics and utilize them for effective alerting purposes. This will enable us to proactively detect and address any potential issues, ensuring optimal performance, availability, and reliability of the database environment.

2.3.1 Read query throughput and performance

Analyzing the metrics related to read query throughput and performance can provide insights into the efficiency of the database in handling read queries.

- **Sequential scans vs. index scans:** If the database is regularly performing more sequential scans over time, its performance could be improved by creating an index on data that is frequently accessed. Optimizing the usage of sequential scans and index scans can lead to improved query performance, reduced query execution times, and overall better database performance.
- **rows fetched and rows returned:** PostgreSQL tracks tup_returned as the number of rows read or scanned during query execution, regardless of whether those rows were actually returned to the client. On the other hand, tup_fetched, or "rows fetched", is the metric that counts the number of rows that contained data which was actually needed to execute the query. By monitoring and analyzing the ratio of rows fetched to rows returned, we can gain insights into how efficiently the database is processing read queries. If there is a significant discrepancy between these metrics, further analysis may be required to identify and optimize queries or table structures that are causing unnecessary row scans. This can help improve the performance of read queries and ensure efficient database operations.

2.3.2 Write query throughput and performance

- **Rows inserted, updated, and deleted:** Monitoring the number of rows inserted, updated, and deleted can help in giving an idea of what types of write queries the database is serving. For instance, if we observe a high rate of updated and deleted rows, it raises a red flag and prompts us to closely monitor the number of dead rows in the database. If the number of dead rows is consistently rising, it can negatively impact query performance and slow down the database. By regularly monitoring and analyzing the number of rows inserted, updated, and deleted, as well as the presence of dead rows, we can proactively identify and address any issues related to write queries.
- **Concurrent operations:** PostgreSQL’s statistics collector provides valuable insights into concurrent operations, allowing us to assess the performance and efficiency of the database in handling multiple operations simultaneously. Metrics such as the number of concurrent connections, lock contention, and wait events provide crucial information on the database’s ability to handle concurrent queries without experiencing performance degradation or bottlenecks. By monitoring these concurrent operation metrics, we can identify any potential issues related to contention for resources, such as locks, which can impact the performance and responsiveness of the database

2.3.3 Resource utilization

Similar to any other sophisticated database system, PostgreSQL utilizes various system resources such as CPU, disk, memory, and network bandwidth to perform its operations efficiently. By diligently monitoring these critical system-level metrics, we can ensure that PostgreSQL has the necessary resources at its disposal to swiftly respond to queries and effectively update data across its tables and indexes. Additionally, PostgreSQL also maintains its own comprehensive set of metrics to monitor its internal resource utilization, encompassing aspects such as connections, shared buffer usage, and disk utilization. In the following sections, we have explained these metrics in greater detail which enables us to better understand and optimize the performance of PostgreSQL.

314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330

331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346

347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363

- **Connections:** When a client requests a connection to the PostgreSQL primary server process, a new process is forked to handle that connection. PostgreSQL establishes a connection limit that determines the maximum number of concurrent connections that can be opened to the backend at any given time. It's important to note that the actual maximum number of connections may be further constrained by the limits set by the operating system.

In scenarios with high concurrency, where there is a large number of concurrent connections, implementing a connection pool such as PgBouncer can be beneficial. The connection pool acts as an intermediary between the applications and the PostgreSQL backends, effectively distributing the direct connections to the primary server.

If we notice that the number of active connections consistently approaches the maximum connection limit, it could be an indication that applications are creating new connections for each request instead of reusing existing connections. This may be due to long-running queries or inefficient connection management. Implementing a connection pool can help mitigate this issue by ensuring that idle connections are consistently reused instead of burdening the primary server with frequent opening and closing of connections.

To further optimize connection management, we set the value for `idle_in_transaction_session_timeout`. This parameter instructs PostgreSQL to automatically close connections that remain idle for a specified period of time. By default, this value is set to 0, which means that this feature is disabled. Adjusting this parameter can help improve resource utilization and connection management in PostgreSQL.

- **Shared Buffer Usage:** When PostgreSQL accesses data, it first checks if the data is already present in the shared buffer cache or the operating system (OS) cache to avoid disk I/O. If the data is not cached, it needs to be retrieved from disk, and it is then cached both in the OS cache and the database's shared buffer cache to optimize future queries. This means that some data may be cached in multiple places

simultaneously. To optimize this caching process, PostgreSQL recommends limiting the `shared_buffers` parameter, which controls the size of the shared buffer cache, to around 25 percent of the available OS memory.

It's important to note that `pg_statio`, which provides statistics related to the shared buffer cache, does not account for hits in the OS cache. Therefore, if the cache hit rate appears low based solely on `pg_statio` statistics, it's crucial to remember that it does not capture hits in the OS cache. To gain a comprehensive understanding of PostgreSQL's actual memory usage, it's recommended to supplement `pg_statio` statistics with metrics related to I/O and memory utilization from the OS kernel. This holistic approach will provide a more accurate picture of PostgreSQL's performance and resource utilization.

- **Disk and index usage:** PostgreSQL has built-in mechanisms to collect statistics that provide insights into the size of tables and indexes over time. These statistics are invaluable for tracking changes in query performance as tables and indexes grow in size. As data volume increases, query execution times may also lengthen, and indexes may require more disk space. This may necessitate scaling up the disk space of the instance, partitioning data to optimize storage, or reevaluating the indexing strategy.

Furthermore, monitoring table and index size can help identify unexpected growth, which may indicate issues with VACUUM operations not running as intended. Properly functioning VACUUMs are crucial for maintaining optimal performance in PostgreSQL. Regular monitoring and analysis of table and index size statistics can provide valuable insights into the health of the database and help identify potential areas for improvement to ensure efficient query performance and resource utilization. Tracking replication delay over time can provide valuable insights into the consistency of data updates across replica servers. A low replication delay indicates that the standby or replica servers are keeping up with the updates from the primary server in near real-time, ensuring that the data is replicated accurately and promptly. On the

other hand, a high replication delay can indicate potential issues, such as network latency, resource constraints, or configuration problems, that may affect the ability of the replica servers to catch up with the updates from the primary server.

2.3.4 Replication and reliability

- **WAL replication in PostgreSQL:** PostgreSQL ensures data integrity and performance by utilizing a write-ahead log (WAL) mechanism for handling data writes and updates. When a transaction is executed, PostgreSQL records the transaction in the WAL, rather than immediately updating the actual data page/block on disk. This approach allows for efficient data updates without sacrificing data reliability. After logging the transaction in the WAL, PostgreSQL checks if the affected data block is already in memory. If the block is in memory, it is updated in memory, and the block is marked as a "dirty page" to indicate that it has been modified. This optimization helps to minimize disk I/O and improve performance, as updates can be handled in memory without immediately writing them to disk.

- **Replication delay:** Monitoring replication delay is an essential part of PostgreSQL replication and reliability monitoring. Replication delay is measured as the time difference between the last write-ahead log (WAL) update received from the primary server and the last WAL update applied or replayed on disk on the standby or replica server. Collecting and graphing replication delay metrics over time can help identify trends and patterns, allowing for proactive monitoring and troubleshooting. For example, sudden spikes in replication delay may indicate a network issue or a heavy load on the replica server, while a persistent increase in replication delay may indicate a need for tuning or optimization in the replica server configuration.

3 Results/Screenshots

We have made three tools to provide different kinds of analytics, Following are the details for the same.

3.1 Python CLI's app integrated with Grafana

Python is used to connect to the database and fetch different metrics. These metrics are then streamed at a UDP socket, which can be used by users to integrate with different monitoring tools, currently, we have integrated it with Grafana: A multi-platform open-source analytics and interactive visualization web application.

[illegible]

Figure 1: Raw metric streamed over UDP socket

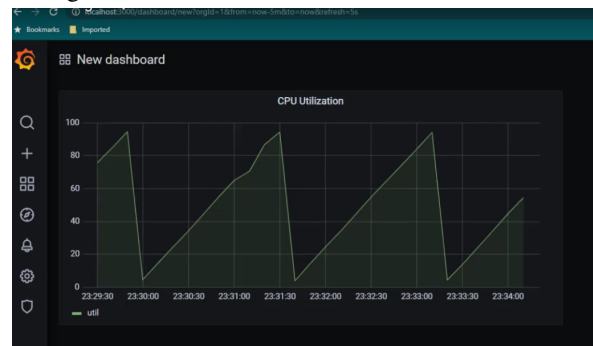


Figure 2: Grafana integrated dashboard

3.2 Python GUI App

We have made an app that provides real-time query analysis and system resource usage in GUI.

```
PostgreSQL 15.2 - rsb-rn - postgresql/var/run/postgresql/152/postgres - Ref.: 2s - Duration mode: query
[Global: 15.2 - rsb-rn and 25 minutes uptime, 29,100 slots active - 86% growth, 100,000 cache hit ratio
15.2 - rsb-rn and 25 minutes uptime, 29,100 slots active - 86% growth, 100,000 cache hit ratio
Activity: 15.2s, 1 inserts/s, 0 updates/s, 0 deletes/s, 933 tuples returned/s, 0 temp files, 00 temp size
Worker processes: 0/8 total, 0/6 logical workers, 0/8 parallel workers
Replication: 0/1 active, 0/0 standby workers, 0/8 total, 0/0 standby receivers, 0/8 repl. slots
Mem.: 7.600 total, 369.300 (+75%) free, 5.626 (73.94%) used, 1.626 (21.32%) buffered+cache
Swap: 22.700 total, 70.488 (60.99%) free, 2.716 (12.12%) used
IO: Max ops: 100/s, 0/s - 100/s, 0/s - 100/s, 0/s - 100/s, 0/s - 100/s, 0/s - 100/s
Load average: 1.89 1.76 1.57
```

RUNNING QUEUES								
PID	DATABASE	APP	USER	CLIENT CPU%	MEM READ/s	WRITE/s	TIME%	Waiting
1	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
2	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
3	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
4	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
5	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
6	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
7	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
8	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
9	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
10	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
11	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
12	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
13	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
14	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
15	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
16	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
17	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
18	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
19	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
20	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
21	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
22	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
23	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
24	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
25	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
26	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
27	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
28	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
29	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
30	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
31	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
32	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
33	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
34	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
35	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
36	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
37	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
38	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
39	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
40	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
41	postgres	postgres	postgres	0.00	0.00	0.00	0.00	0.00
42	postgres	postgres	postgres	0.00	0.00			

Figure 3: System resource usage

```
PostgreSQL 15.2 - yashraj-zenbook-ubuntu - postgres@127.0.0.1:5432/postgres - Ref.: 2s
Size: 94.79M - 00/0s - TPS: 1 - Active connections: 4 - Duration: model query
WARNING:
PID DATABASE APP USER CLIENT TIME+ STATE Query
151592 local 00:00:49
151590 active select + from
151588 local 00:00:14
151588 active select + from
151589 'InvoiceLine', 'MediaTypes', 'Track', 'Playlist';
00:00:19 active
151589 'Album', 'InvoiceLine', 'MediaTypes', 'Track';
00:00:19 active select + from 'Track'
```

Figure 4: Live query analysis

3.3 GO CLI app to provide in-depth table and overall database analysis

This app provides two reports, whole DB analysis, and individual table analysis. Here is the sample output run on our HMS project's database. Click [here](#) to view the reports: [data](#)

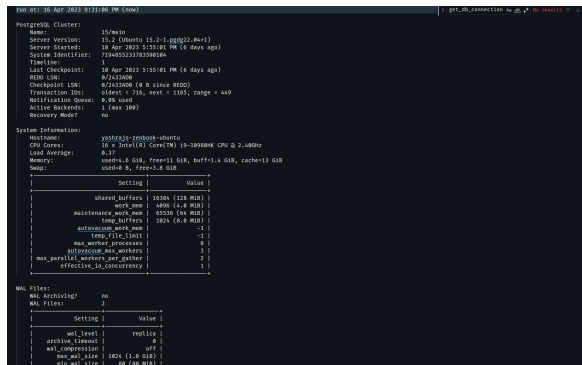


Figure 5: detailed in-depth analysis

References

- datadoghq.com/blog/postgresql-monitoring-tools
Collecting metrics with PostgreSQL monitoring tools
- datadoghq.com/blog/postgresql-monitoring-tools
Collecting metrics with PostgreSQL monitoring tools
- datadoghq.com/blog/postgresql-monitoring-tools
Key metrics for PostgreSQL monitoring
- wiki.postgresql.org/wiki/Monitoring
Monitoring
- postgresql.org/docs/current/monitoring
Monitoring Database Activity
- docs.digitalocean.com/products/databases/postgresql
How to Monitor PostgreSQL Database Performance
- severalnines.com/blog/key-things-monitor-postgresql-analyzing-your-workload
Key Things to Monitor in PostgreSQL – Analyzing Your Workload
- fatdba.com/2021/03/24/how-to-monitor-your-postgresql-database-using-grafana
How to monitor your PostgreSQL database using Grafana, Prometheus postgres_exporter
- grafana.com/solutions/postgresql/monitor
Monitor PostgreSQL easily with Grafana