Analytics — NODES (Paths, Code, Cognitive Memory)

Artifact: cognition-module()analytics-nodes.pdf

Scope: Analytics module only · Evidence-only · Deterministic · Minimal-change semantics

Node: CallsDurationSummary (calls-duration-summary)

1) Paths

```
call-metrics-analytics/
  src/dashboard/calls-duration-summary/
    schema.graphql
    resolver.ts
    service.ts
    pipeline/
      index.ts
    types/
      index.ts
```

Shared Types (import once)

```
// call-metrics-analytics/src/dashboard/shared/types/index.ts
export type DailyDataInput = {
  startDate: string;
  endDate: string;
  clinicTimezone: string;
};
```

2) Code — schema.graphql

```
# call-metrics-analytics/src/dashboard/calls-duration-summary/schema.graphql
# Minimal, copy-pasteable SDL consistent with Analytics conventions.
# Follows DailyDataInput signature (startDate, endDate, clinicTimezone).
# Does not change established semantics; exact fields may be extended in implementation.

input CallsDurationSummaryRangeInput {
  startDate: String!
  endDate: String!
  clinicTimezone: String!
}

type CallsDurationSummaryResult {
  # Example fields — keep minimal & extend in resolver/service as needed.
  days: [String!]!
  data: [Float!]!
  # Optionally include meta if already standard in your Analytics layer.
  # meta: JSON
}

extend type Query {
  calls_duration_summary(range: CallsDurationSummaryRangeInput!): CallsDurationSummaryResult!
}
```

2) Code — pipeline/index.ts

```
// call-metrics-analytics/src/dashboard/calls-duration-summary/pipeline/index.ts
import { DailyDataInput } from "../../shared/types";
```

```
export function generatePipeline({ startDate, endDate, clinicTimezone }: DailyDataInput) {
  // Evidence-only baseline: follow the known pattern with $match on callStatus and time range.
  // Replace collection stages as needed for this node's aggregation.

  return [
    {
      $match: {
        $expr: {
          $and: [
            { $ne: ["$callStatus", "ongoing-bot"] },
            { $ne: ["$callStatus", "ongoing-agent"] },
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] }
          ]
        }
      }
    },
    // TODO: Add node-specific stages here (grouping, bucketing, projections).
    // Keep naming aligned with existing Analytics conventions.
  ];
}
```

2) Code — service.ts

```
// call-metrics-analytics/src/dashboard/calls-duration-summary/service.ts
import type { DailyDataInput } from "../shared/types";
import { generatePipeline } from "./pipeline";
import { MongoClient } from "mongodb";

export class CallsDurationSummaryService {
  constructor(private readonly client: MongoClient, private readonly dbName: string) { }

  async getCallsDurationSummary(input: DailyDataInput) {
    const pipeline = generatePipeline(input);
    const db = this.client.db(this.dbName);
    // Replace 'callsInformation' if your collection differs.
    const cursor = db.collection("callsInformation").aggregate(pipeline, { allowDiskUse: true });
    const result = await cursor.toArray();

    // Normalize to the minimal shape declared in schema.graphql
    // NOTE: Keep this minimal; extend only with evidence-backed fields.
    const days: string[] = result[0]?.days ?? [];
    const data: number[] = result[0]?.data ?? [];
    return { days, data };
  }
}
```

2) Code — resolver.ts

```
// call-metrics-analytics/src/dashboard/calls-duration-summary/resolver.ts
import { Resolver, Query, Args } from "@nestjs/graphql";
import { CallsDurationSummaryService } from "./service";

@Resolver()
export class CallsDurationSummaryResolver {
  constructor(private readonly service: CallsDurationSummaryService) { }

  @Query(() => CallsDurationSummaryResult, { name: "calls_duration_summary" })
  async queryCallsDurationSummary(
    @Args("range") range: { startDate: string; endDate: string; clinicTimezone: string }
  ) {
    return this.service.getCallsDurationSummary}(range);
  }
}
```

2) Code — types/index.ts

```
// call-metrics-analytics/src/dashboard/calls-duration-summary/types/index.ts
export type CallsDurationSummaryResult = {
  days: string[];
  data: number[];
  // meta?: Record<string, unknown>;
};
```

3) Minimal Cognitive Memory

```
{
  "cognitive_node": {
    "id": "analytics.calls-duration-summary.v1",
    "kind": "AnalyticsNode",
    "title": "CallsDurationSummary",
    "purpose": "Expose calls-duration-summary metrics via GraphQL with minimal, stable schema; compu
    "contracts": {
      "inputs": ["DailyDataInput(startDate, endDate, clinicTimezone)"],
      "outputs": ["CallsDurationSummaryResult"],
      "non_functionals": [
        "Deterministic",
        "Evidence-only",
        "Consistent naming and folder structure"
      ]
    }
  },
  "graph_patch": {
    "nodes": [
      { "id": "analytics.calls-duration-summary.v1", "kind": "AnalyticsNode" }
    ],
    "edges": [
      { "from": "analytics.calls-duration-summary.v1", "to": "cognition-module()analytics.pdf", "kin
      { "from": "analytics.calls-duration-summary.v1", "to": "cognitive-modules()api-gateway.pdf", "
    ]
  }
}
```

Node: HandlingOverview (handling-overview)

1) Paths

```
call-metrics-analytics/
  src/dashboard/handling-overview/
    schema.graphql
    resolver.ts
    service.ts
    pipeline/
      index.ts
    types/
      index.ts
```

2) Code — schema.graphql

```
# call-metrics-analytics/src/dashboard/handling-overview/schema.graphql
# Minimal, copy-pasteable SDL consistent with Analytics conventions.
# Follows DailyDataInput signature (startDate, endDate, clinicTimezone).
# Does not change established semantics; exact fields may be extended in implementation.

input HandlingOverviewRangeInput {
  startDate: String!
  endDate: String!
  clinicTimezone: String!
}

type HandlingOverviewResult {
  # Example fields — keep minimal & extend in resolver/service as needed.
  days: [String!]!
  data: [Float!]!
  # Optionally include meta if already standard in your Analytics layer.
  # meta: JSON
}

extend type Query {
  handling_overview(range: HandlingOverviewRangeInput!): HandlingOverviewResult!
}
```

2) Code — pipeline/index.ts

```
// call-metrics-analytics/src/dashboard/handling-overview/pipeline/index.ts
import { DailyDataInput } from "../../shared/types";

export function generatePipeline({ startDate, endDate, clinicTimezone }: DailyDataInput) {
  // Evidence-only baseline: follow the known pattern with $match on callStatus and time range.
  // Replace collection stages as needed for this node's aggregation.

  return [
    {
      $match: {
        $expr: {
          $and: [
            { $ne: ["$callStatus", "ongoing-bot"] },
            { $ne: ["$callStatus", "ongoing-agent"] },
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] }
          ]
        }
      }
    },
    // TODO: Add node-specific stages here (grouping, bucketing, projections).
    // Keep naming aligned with existing Analytics conventions.
  ];
}
```

2) Code — service.ts

```
// call-metrics-analytics/src/dashboard/handling-overview/service.ts
import type { DailyDataInput } from "../shared/types";
import { generatePipeline } from "./pipeline";
import { MongoClient } from "mongodb";

export class HandlingOverviewService {
  constructor(private readonly client: MongoClient, private readonly dbName: string) { }

  async getHandlingOverview(input: DailyDataInput) {
    const pipeline = generatePipeline(input);
    const db = this.client.db(this.dbName);
    // Replace 'callsInformation' if your collection differs.
    const cursor = db.collection("callsInformation").aggregate(pipeline, { allowDiskUse: true });
    const result = await cursor.toArray();

    // Normalize to the minimal shape declared in schema.graphql
    // NOTE: Keep this minimal; extend only with evidence-backed fields.
    const days: string[] = result[0]?.days ?? [];
    const data: number[] = result[0]?.data ?? [];
    return { days, data };
  }
}
```

2) Code — resolver.ts

```
// call-metrics-analytics/src/dashboard/handling-overview/resolver.ts
import { Resolver, Query, Args } from "@nestjs/graphql";
import { HandlingOverviewService } from "./service";

@Resolver()
export class HandlingOverviewResolver {
  constructor(private readonly service: HandlingOverviewService) { }

  @Query(() => HandlingOverviewResult, { name: "handling_overview" })
  async queryHandlingOverview(
    @Args("range") range: { startDate: string; endDate: string; clinicTimezone: string }
  ) {
    return this.service.getHandlingOverview}(range);
  }
}
```

2) Code — types/index.ts

```
// call-metrics-analytics/src/dashboard/handling-overview/types/index.ts
export type HandlingOverviewResult = {
  days: string[];
  data: number[];
  // meta?: Record<string, unknown>;
};
```

3) Minimal Cognitive Memory

```
{
  "cognitive_node": {
    "id": "analytics.handling-overview.v1",
    "kind": "AnalyticsNode",
    "title": "HandlingOverview",
    "purpose": "Expose handling-overview metrics via GraphQL with minimal, stable schema; compute vi
    "contracts": {
      "inputs": ["DailyDataInput(startDate, endDate, clinicTimezone)"],
      "outputs": ["HandlingOverviewResult"],
      "non_functionals": [
        "Deterministic",
        "Evidence-only",
        "Consistent naming and folder structure"
      ]
    }
  },
  "graph_patch": {
    "nodes": [
      { "id": "analytics.handling-overview.v1", "kind": "AnalyticsNode" }
    ],
    "edges": [
      { "from": "analytics.handling-overview.v1", "to": "cognition-module()analytics.pdf", "kind": "
      { "from": "analytics.handling-overview.v1", "to": "cognitive-modules()api-gateway.pdf", "kind"
    ]
  }
}
```

Node: CallVolume (call-volume)

1) Paths

```
call-metrics-analytics/
  src/dashboard/call-volume/
    schema.graphql
    resolver.ts
    service.ts
    pipeline/
      index.ts
    types/
      index.ts
```

2) Code — schema.graphql

```
# call-metrics-analytics/src/dashboard/call-volume/schema.graphql
# Minimal, copy-pasteable SDL consistent with Analytics conventions.
# Follows DailyDataInput signature (startDate, endDate, clinicTimezone).
# Does not change established semantics; exact fields may be extended in implementation.

input CallVolumeRangeInput {
  startDate: String!
  endDate: String!
  clinicTimezone: String!
}

type CallVolumeResult {
  # Example fields — keep minimal & extend in resolver/service as needed.
  days: [String!]!
  data: [Float!]!
  # Optionally include meta if already standard in your Analytics layer.
  # meta: JSON
}

extend type Query {
  call_volume(range: CallVolumeRangeInput!): CallVolumeResult!
}
```

2) Code — pipeline/index.ts

```
// call-metrics-analytics/src/dashboard/call-volume/pipeline/index.ts
import { DailyDataInput } from "../../shared/types";

export function generatePipeline({ startDate, endDate, clinicTimezone }: DailyDataInput) {
  // Evidence-only baseline: follow the known pattern with $match on callStatus and time range.
  // Replace collection stages as needed for this node's aggregation.

  return [
    {
      $match: {
        $expr: {
          $and: [
            { $ne: ["$callStatus", "ongoing-bot"] },
            { $ne: ["$callStatus", "ongoing-agent"] },
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] }
          ]
        }
      }
    },
    // TODO: Add node-specific stages here (grouping, bucketing, projections).
    // Keep naming aligned with existing Analytics conventions.
  ];
}
```

2) Code — service.ts

```
// call-metrics-analytics/src/dashboard/call-volume/service.ts
import type { DailyDataInput } from "../shared/types";
import { generatePipeline } from "./pipeline";
import { MongoClient } from "mongodb";

export class CallVolumeService {
  constructor(private readonly client: MongoClient, private readonly dbName: string) { }

  async getCallVolume(input: DailyDataInput) {
    const pipeline = generatePipeline(input);
    const db = this.client.db(this.dbName);
    // Replace 'callsInformation' if your collection differs.
    const cursor = db.collection("callsInformation").aggregate(pipeline, { allowDiskUse: true });
    const result = await cursor.toArray();

    // Normalize to the minimal shape declared in schema.graphql
    // NOTE: Keep this minimal; extend only with evidence-backed fields.
    const days: string[] = result[0]?.days ?? [];
    const data: number[] = result[0]?.data ?? [];
    return { days, data };
  }
}
```

2) Code — resolver.ts

```
// call-metrics-analytics/src/dashboard/call-volume/resolver.ts
import { Resolver, Query, Args } from "@nestjs/graphql";
import { CallVolumeService } from "./service";

@Resolver()
export class CallVolumeResolver {
  constructor(private readonly service: CallVolumeService) { }

  @Query(() => CallVolumeResult, { name: "call_volume" })
  async queryCallVolume(
    @Args("range") range: { startDate: string; endDate: string; clinicTimezone: string }
  ) {
    return this.service.getCallVolume}(range);
  }
}
```

2) Code — types/index.ts

```
// call-metrics-analytics/src/dashboard/call-volume/types/index.ts
export type CallVolumeResult = {
  days: string[];
  data: number[];
  // meta?: Record<string, unknown>;
};
```

3) Minimal Cognitive Memory

```
{
  "cognitive_node": {
    "id": "analytics.call-volume.v1",
    "kind": "AnalyticsNode",
    "title": "CallVolume",
    "purpose": "Expose call-volume metrics via GraphQL with minimal, stable schema; compute via Mong
    "contracts": {
      "inputs": ["DailyDataInput(startDate, endDate, clinicTimezone)"],
      "outputs": ["CallVolumeResult"],
      "non_functionals": [
        "Deterministic",
        "Evidence-only",
        "Consistent naming and folder structure"
      ]
    }
  },
  "graph_patch": {
    "nodes": [
      { "id": "analytics.call-volume.v1", "kind": "AnalyticsNode" }
    ],
    "edges": [
      { "from": "analytics.call-volume.v1", "to": "cognition-module()analytics.pdf", "kind": "PART_(
      { "from": "analytics.call-volume.v1", "to": "cognitive-modules()api-gateway.pdf", "kind": "COM
    ]
  }
}
```

Node: AiOperationBreakdown (ai-operation-breakdown)

1) Paths

```
call-metrics-analytics/
  src/dashboard/ai-operation-breakdown/
    schema.graphql
    resolver.ts
    service.ts
    pipeline/
      index.ts
    types/
      index.ts
```

2) Code — schema.graphql

```
# call-metrics-analytics/src/dashboard/ai-operation-breakdown/schema.graphql
# Minimal, copy-pasteable SDL consistent with Analytics conventions.
# Follows DailyDataInput signature (startDate, endDate, clinicTimezone).
# Does not change established semantics; exact fields may be extended in implementation.

input AiOperationBreakdownRangeInput {
  startDate: String!
  endDate: String!
  clinicTimezone: String!
}

type AiOperationBreakdownResult {
  # Example fields — keep minimal & extend in resolver/service as needed.
  days: [String!]!
  data: [Float!]!
  # Optionally include meta if already standard in your Analytics layer.
  # meta: JSON
}

extend type Query {
  ai_operation_breakdown(range: AiOperationBreakdownRangeInput!): AiOperationBreakdownResult!
}
```

2) Code — pipeline/index.ts

```
// call-metrics-analytics/src/dashboard/ai-operation-breakdown/pipeline/index.ts
import { DailyDataInput } from "../../shared/types";

export function generatePipeline({ startDate, endDate, clinicTimezone }: DailyDataInput) {
  // Evidence-only baseline: follow the known pattern with $match on callStatus and time range.
  // Replace collection stages as needed for this node's aggregation.

  return [
    {
      $match: {
        $expr: {
          $and: [
            { $ne: ["$callStatus", "ongoing-bot"] },
            { $ne: ["$callStatus", "ongoing-agent"] },
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] }
          ]
        }
      }
    },
    // TODO: Add node-specific stages here (grouping, bucketing, projections).
    // Keep naming aligned with existing Analytics conventions.
  ];
}
```

2) Code — service.ts

```
// call-metrics-analytics/src/dashboard/ai-operation-breakdown/service.ts
import type { DailyDataInput } from "../shared/types";
import { generatePipeline } from "./pipeline";
import { MongoClient } from "mongodb";

export class AiOperationBreakdownService {
  constructor(private readonly client: MongoClient, private readonly dbName: string) { }

  async getAiOperationBreakdown(input: DailyDataInput) {
    const pipeline = generatePipeline(input);
    const db = this.client.db(this.dbName);
    // Replace 'callsInformation' if your collection differs.
    const cursor = db.collection("callsInformation").aggregate(pipeline, { allowDiskUse: true });
    const result = await cursor.toArray();

    // Normalize to the minimal shape declared in schema.graphql
    // NOTE: Keep this minimal; extend only with evidence-backed fields.
    const days: string[] = result[0]?.days ?? [];
    const data: number[] = result[0]?.data ?? [];
    return { days, data };
  }
}
```

2) Code — resolver.ts

```
// call-metrics-analytics/src/dashboard/ai-operation-breakdown/resolver.ts
import { Resolver, Query, Args } from "@nestjs/graphql";
import { AiOperationBreakdownService } from "./service";

@Resolver()
export class AiOperationBreakdownResolver {
  constructor(private readonly service: AiOperationBreakdownService) { }

  @Query(() => AiOperationBreakdownResult, { name: "ai_operation_breakdown" })
  async queryAiOperationBreakdown(
    @Args("range") range: { startDate: string; endDate: string; clinicTimezone: string }
  ) {
    return this.service.getAiOperationBreakdown}(range);
  }
}
```

2) Code — types/index.ts

```
// call-metrics-analytics/src/dashboard/ai-operation-breakdown/types/index.ts
export type AiOperationBreakdownResult = {
  days: string[];
  data: number[];
  // meta?: Record<string, unknown>;
};
```

3) Minimal Cognitive Memory

```
{
  "cognitive_node": {
    "id": "analytics.ai-operation-breakdown.v1",
    "kind": "AnalyticsNode",
    "title": "AiOperationBreakdown",
    "purpose": "Expose ai-operation-breakdown metrics via GraphQL with minimal, stable schema; compu
    "contracts": {
      "inputs": ["DailyDataInput(startDate, endDate, clinicTimezone)"],
      "outputs": ["AiOperationBreakdownResult"],
      "non_functionals": [
        "Deterministic",
        "Evidence-only",
        "Consistent naming and folder structure"
      ]
    }
  },
  "graph_patch": {
    "nodes": [
      { "id": "analytics.ai-operation-breakdown.v1", "kind": "AnalyticsNode" }
    ],
    "edges": [
      { "from": "analytics.ai-operation-breakdown.v1", "to": "cognition-module()analytics.pdf", "ki
      { "from": "analytics.ai-operation-breakdown.v1", "to": "cognitive-modules()api-gateway.pdf", "
    ]
  }
}
```

Node: CallFrequencyOutcome (call-frequency-outcome)

1) Paths

```
call-metrics-analytics/
  src/dashboard/call-frequency-outcome/
    schema.graphql
    resolver.ts
    service.ts
    pipeline/
      index.ts
    types/
      index.ts
```

2) Code — schema.graphql

```graphql
# call-metrics-analytics/src/dashboard/call-frequency-outcome/schema.graphql
# Minimal, copy-pasteable SDL consistent with Analytics conventions.
# Follows DailyDataInput signature (startDate, endDate, clinicTimezone).
# Does not change established semantics; exact fields may be extended in implementation.

input CallFrequencyOutcomeRangeInput {
  startDate: String!
  endDate: String!
  clinicTimezone: String!
}

type CallFrequencyOutcomeResult {
  # Example fields — keep minimal & extend in resolver/service as needed.
  days: [String!]!
  data: [Float!]!
  # Optionally include meta if already standard in your Analytics layer.
  # meta: JSON
}

extend type Query {
  call_frequency_outcome(range: CallFrequencyOutcomeRangeInput!): CallFrequencyOutcomeResult!
}
```

2) Code — pipeline/index.ts

```ts
// call-metrics-analytics/src/dashboard/call-frequency-outcome/pipeline/index.ts
import { DailyDataInput } from "../../shared/types";

export function generatePipeline({ startDate, endDate, clinicTimezone }: DailyDataInput) {
  // Evidence-only baseline: follow the known pattern with $match on callStatus and time range.
  // Replace collection stages as needed for this node's aggregation.

  return [
    {
      $match: {
        $expr: {
          $and: [
            { $ne: ["$callStatus", "ongoing-bot"] },
            { $ne: ["$callStatus", "ongoing-agent"] },
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] }
          ]
        }
      }
    },
    // TODO: Add node-specific stages here (grouping, bucketing, projections).
    // Keep naming aligned with existing Analytics conventions.
  ];
}
```

2) Code — service.ts

```
// call-metrics-analytics/src/dashboard/call-frequency-outcome/service.ts
import type { DailyDataInput } from "../shared/types";
import { generatePipeline } from "./pipeline";
import { MongoClient } from "mongodb";

export class CallFrequencyOutcomeService {
  constructor(private readonly client: MongoClient, private readonly dbName: string) { }

  async getCallFrequencyOutcome(input: DailyDataInput) {
    const pipeline = generatePipeline(input);
    const db = this.client.db(this.dbName);
    // Replace 'callsInformation' if your collection differs.
    const cursor = db.collection("callsInformation").aggregate(pipeline, { allowDiskUse: true });
    const result = await cursor.toArray();

    // Normalize to the minimal shape declared in schema.graphql
    // NOTE: Keep this minimal; extend only with evidence-backed fields.
    const days: string[] = result[0]?.days ?? [];
    const data: number[] = result[0]?.data ?? [];
    return { days, data };
  }
}
```

2) Code — resolver.ts

```
// call-metrics-analytics/src/dashboard/call-frequency-outcome/resolver.ts
import { Resolver, Query, Args } from "@nestjs/graphql";
import { CallFrequencyOutcomeService } from "./service";

@Resolver()
export class CallFrequencyOutcomeResolver {
  constructor(private readonly service: CallFrequencyOutcomeService) { }

  @Query(() => CallFrequencyOutcomeResult, { name: "call_frequency_outcome" })
  async queryCallFrequencyOutcome(
    @Args("range") range: { startDate: string; endDate: string; clinicTimezone: string }
  ) {
    return this.service.getCallFrequencyOutcome}(range);
  }
}
```

2) Code — types/index.ts

```
// call-metrics-analytics/src/dashboard/call-frequency-outcome/types/index.ts
export type CallFrequencyOutcomeResult = {
  days: string[];
  data: number[];
  // meta?: Record<string, unknown>;
};
```

3) Minimal Cognitive Memory

```
{
  "cognitive_node": {
    "id": "analytics.call-frequency-outcome.v1",
    "kind": "AnalyticsNode",
    "title": "CallFrequencyOutcome",
    "purpose": "Expose call-frequency-outcome metrics via GraphQL with minimal, stable schema; compu
    "contracts": {
      "inputs": ["DailyDataInput(startDate, endDate, clinicTimezone)"],
      "outputs": ["CallFrequencyOutcomeResult"],
      "non_functionals": [
        "Deterministic",
        "Evidence-only",
        "Consistent naming and folder structure"
      ]
    }
  },
  "graph_patch": {
    "nodes": [
      { "id": "analytics.call-frequency-outcome.v1", "kind": "AnalyticsNode" }
    ],
    "edges": [
      { "from": "analytics.call-frequency-outcome.v1", "to": "cognition-module()analytics.pdf", "kin
      { "from": "analytics.call-frequency-outcome.v1", "to": "cognitive-modules()api-gateway.pdf", "
    ]
  }
}
```

Node: MedianCallDuration (median-call-duration)

1) Paths

```
call-metrics-analytics/
  src/dashboard/median-call-duration/
    schema.graphql
    resolver.ts
    service.ts
    pipeline/
      index.ts
    types/
      index.ts
```

2) Code — schema.graphql

```
# call-metrics-analytics/src/dashboard/median-call-duration/schema.graphql
# Minimal, copy-pasteable SDL consistent with Analytics conventions.
# Follows DailyDataInput signature (startDate, endDate, clinicTimezone).
# Does not change established semantics; exact fields may be extended in implementation.

input MedianCallDurationRangeInput {
  startDate: String!
  endDate: String!
  clinicTimezone: String!
}

type MedianCallDurationResult {
  # Example fields — keep minimal & extend in resolver/service as needed.
  days: [String!]!
  data: [Float!]!
  # Optionally include meta if already standard in your Analytics layer.
  # meta: JSON
}

extend type Query {
  median_call_duration(range: MedianCallDurationRangeInput!): MedianCallDurationResult!
}
```

2) Code — pipeline/index.ts

```
// call-metrics-analytics/src/dashboard/median-call-duration/pipeline/index.ts
import { DailyDataInput } from "../../shared/types";

export function generatePipeline({ startDate, endDate, clinicTimezone }: DailyDataInput) {
  // Evidence-only baseline: follow the known pattern with $match on callStatus and time range.
  // Replace collection stages as needed for this node's aggregation.

  return [
    {
      $match: {
        $expr: {
          $and: [
            { $ne: ["$callStatus", "ongoing-bot"] },
            { $ne: ["$callStatus", "ongoing-agent"] },
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] }
          ]
        }
      }
    },
    // TODO: Add node-specific stages here (grouping, bucketing, projections).
    // Keep naming aligned with existing Analytics conventions.
  ];
}
```

2) Code — service.ts

```
// call-metrics-analytics/src/dashboard/median-call-duration/service.ts
import type { DailyDataInput } from "../shared/types";
import { generatePipeline } from "./pipeline";
import { MongoClient } from "mongodb";

export class MedianCallDurationService {
  constructor(private readonly client: MongoClient, private readonly dbName: string) { }

  async getMedianCallDuration(input: DailyDataInput) {
    const pipeline = generatePipeline(input);
    const db = this.client.db(this.dbName);
    // Replace 'callsInformation' if your collection differs.
    const cursor = db.collection("callsInformation").aggregate(pipeline, { allowDiskUse: true });
    const result = await cursor.toArray();

    // Normalize to the minimal shape declared in schema.graphql
    // NOTE: Keep this minimal; extend only with evidence-backed fields.
    const days: string[] = result[0]?.days ?? [];
    const data: number[] = result[0]?.data ?? [];
    return { days, data };
  }
}
```

2) Code — resolver.ts

```
// call-metrics-analytics/src/dashboard/median-call-duration/resolver.ts
import { Resolver, Query, Args } from "@nestjs/graphql";
import { MedianCallDurationService } from "./service";

@Resolver()
export class MedianCallDurationResolver {
  constructor(private readonly service: MedianCallDurationService) { }

  @Query(() => MedianCallDurationResult, { name: "median_call_duration" })
  async queryMedianCallDuration(
    @Args("range") range: { startDate: string; endDate: string; clinicTimezone: string }
  ) {
    return this.service.getMedianCallDuration}(range);
  }
}
```

2) Code — types/index.ts

```
// call-metrics-analytics/src/dashboard/median-call-duration/types/index.ts
export type MedianCallDurationResult = {
  days: string[];
  data: number[];
  // meta?: Record<string, unknown>;
};
```

3) Minimal Cognitive Memory

```
{
  "cognitive_node": {
    "id": "analytics.median-call-duration.v1",
    "kind": "AnalyticsNode",
    "title": "MedianCallDuration",
    "purpose": "Expose median-call-duration metrics via GraphQL with minimal, stable schema; compute
    "contracts": {
      "inputs": ["DailyDataInput(startDate, endDate, clinicTimezone)"],
      "outputs": ["MedianCallDurationResult"],
      "non_functionals": [
        "Deterministic",
        "Evidence-only",
        "Consistent naming and folder structure"
      ]
    }
  },
  "graph_patch": {
    "nodes": [
      { "id": "analytics.median-call-duration.v1", "kind": "AnalyticsNode" }
    ],
    "edges": [
      { "from": "analytics.median-call-duration.v1", "to": "cognition-module()analytics.pdf", "kind"
      { "from": "analytics.median-call-duration.v1", "to": "cognitive-modules()api-gateway.pdf", "ki
    ]
  }
}
```

Node: AgentCallTime (agent-call-time)

1) Paths

```
call-metrics-analytics/
  src/dashboard/agent-call-time/
    schema.graphql
    resolver.ts
    service.ts
    pipeline/
      index.ts
    types/
      index.ts
```

2) Code — schema.graphql

```
# call-metrics-analytics/src/dashboard/agent-call-time/schema.graphql
# Minimal, copy-pasteable SDL consistent with Analytics conventions.
# Follows DailyDataInput signature (startDate, endDate, clinicTimezone).
# Does not change established semantics; exact fields may be extended in implementation.

input AgentCallTimeRangeInput {
  startDate: String!
  endDate: String!
  clinicTimezone: String!
}

type AgentCallTimeResult {
  # Example fields — keep minimal & extend in resolver/service as needed.
  days: [String!]!
  data: [Float!]!
  # Optionally include meta if already standard in your Analytics layer.
  # meta: JSON
}

extend type Query {
  agent_call_time(range: AgentCallTimeRangeInput!): AgentCallTimeResult!
}
```

2) Code — pipeline/index.ts

```
// call-metrics-analytics/src/dashboard/agent-call-time/pipeline/index.ts
import { DailyDataInput } from "../../shared/types";

export function generatePipeline({ startDate, endDate, clinicTimezone }: DailyDataInput) {
  // Evidence-only baseline: follow the known pattern with $match on callStatus and time range.
  // Replace collection stages as needed for this node's aggregation.

  return [
    {
      $match: {
        $expr: {
          $and: [
            { $ne: ["$callStatus", "ongoing-bot"] },
            { $ne: ["$callStatus", "ongoing-agent"] },
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] }
          ]
        }
      }
    },
    // TODO: Add node-specific stages here (grouping, bucketing, projections).
    // Keep naming aligned with existing Analytics conventions.
  ];
}
```

2) Code — service.ts

```
// call-metrics-analytics/src/dashboard/agent-call-time/service.ts
import type { DailyDataInput } from "../shared/types";
import { generatePipeline } from "./pipeline";
import { MongoClient } from "mongodb";

export class AgentCallTimeService {
  constructor(private readonly client: MongoClient, private readonly dbName: string) { }

  async getAgentCallTime(input: DailyDataInput) {
    const pipeline = generatePipeline(input);
    const db = this.client.db(this.dbName);
    // Replace 'callsInformation' if your collection differs.
    const cursor = db.collection("callsInformation").aggregate(pipeline, { allowDiskUse: true });
    const result = await cursor.toArray();

    // Normalize to the minimal shape declared in schema.graphql
    // NOTE: Keep this minimal; extend only with evidence-backed fields.
    const days: string[] = result[0]?.days ?? [];
    const data: number[] = result[0]?.data ?? [];
    return { days, data };
  }
}
```

2) Code — resolver.ts

```
// call-metrics-analytics/src/dashboard/agent-call-time/resolver.ts
import { Resolver, Query, Args } from "@nestjs/graphql";
import { AgentCallTimeService } from "./service";

@Resolver()
export class AgentCallTimeResolver {
  constructor(private readonly service: AgentCallTimeService) { }

  @Query(() => AgentCallTimeResult, { name: "agent_call_time" })
  async queryAgentCallTime(
    @Args("range") range: { startDate: string; endDate: string; clinicTimezone: string }
  ) {
    return this.service.getAgentCallTime}(range);
  }
}
```

2) Code — types/index.ts

```
// call-metrics-analytics/src/dashboard/agent-call-time/types/index.ts
export type AgentCallTimeResult = {
  days: string[];
  data: number[];
  // meta?: Record<string, unknown>;
};
```

3) Minimal Cognitive Memory

```
{
  "cognitive_node": {
    "id": "analytics.agent-call-time.v1",
    "kind": "AnalyticsNode",
    "title": "AgentCallTime",
    "purpose": "Expose agent-call-time metrics via GraphQL with minimal, stable schema; compute via
    "contracts": {
      "inputs": ["DailyDataInput(startDate, endDate, clinicTimezone)"],
      "outputs": ["AgentCallTimeResult"],
      "non_functionals": [
        "Deterministic",
        "Evidence-only",
        "Consistent naming and folder structure"
      ]
    }
  },
  "graph_patch": {
    "nodes": [
      { "id": "analytics.agent-call-time.v1", "kind": "AnalyticsNode" }
    ],
    "edges": [
      { "from": "analytics.agent-call-time.v1", "to": "cognition-module()analytics.pdf", "kind": "P
      { "from": "analytics.agent-call-time.v1", "to": "cognitive-modules()api-gateway.pdf", "kind":
    ]
  }
}
```