facade Manual

A lightweight Typst package for creating clean block diagrams with CSS flexbox-like layouts.

Example 1: Simple Vertical Layers

The most basic use case — a vertical stack of layers:

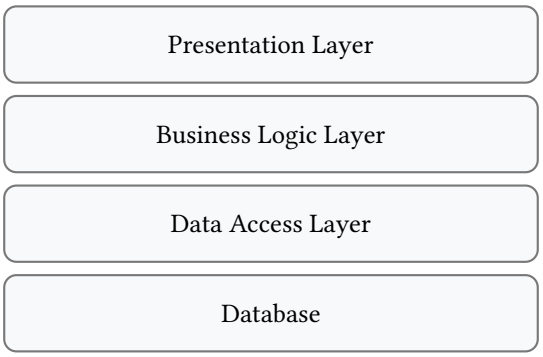

Figure 1: A simple four-tier architecture using vertical layers.

```
#dg-layers(
  dg-rect([Presentation Layer], fill: rgb("#f8f9fa"), width: 200pt),
  dg-rect([Business Logic Layer], fill: rgb("#f8f9fa"), width: 200pt),
  dg-rect([Data Access Layer], fill: rgb("#f8f9fa"), width: 200pt),
  dg-rect([Database], fill: rgb("#f8f9fa"), width: 200pt),
)
```

Example 2: Horizontal Components with Circles

Using `dg-flex` for horizontal layout and `dg-circle` for nodes:

Figure 2: Horizontal row of circular nodes.

```
#dg-flex(
  direction: "row",
  justify: "center",
  gap: 1.2em,
  dg-circle([A], fill: rgb("#e3f2fd")),
  dg-circle([B], fill: rgb("#e8f5e9")),
  dg-circle([C], fill: rgb("#fff3e0")),
  dg-circle([D], fill: rgb("#fce4ec")),
)
```

Example 3: Nested Layout — Services Inside Layers

Combining `dg-layers` with nested `dg-flex`:

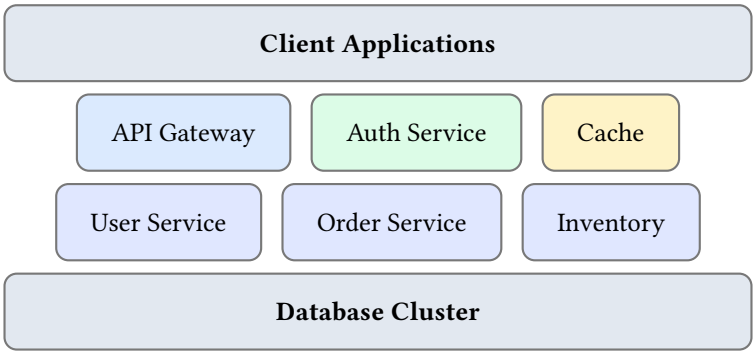

Figure 3: Nested layout with services grouped inside layers.

```
#dg-layers(
  dg-rect([*Client Applications*], fill: rgb("#e2e8f0"), width: 280pt),
  dg-flex(
    direction: "row",
    justify: "center",
    dg-rect([API Gateway], fill: rgb("#dbeafe")),
    dg-rect([Auth Service], fill: rgb("#dcfce7")),
    dg-rect([Cache], fill: rgb("#fef3c7")),
  ),
  // ... more layers
)
```

Example 4: Mixed Shapes

Combining rectangles, circles, and ellipses:

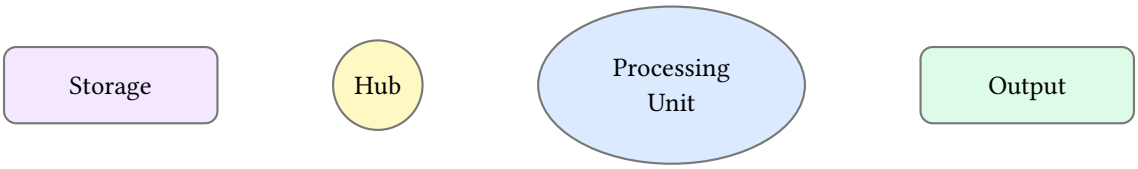

Figure 4: Mixed shapes: rectangles, circles, and ellipses.

```
#dg-flex(
  direction: "row",
  justify: "space-around",
  align-items: "center",
  gap: 1em,
  dg-rect([Storage], fill: rgb("#f3e8ff"), width: 80pt),
  dg-circle([Hub], fill: rgb("#fef9c3")),
  dg-ellipse([Processing Unit], fill: rgb("#dbeafe"), width: 100pt),
  dg-rect([Output], fill: rgb("#dcfce7"), width: 80pt),
)
```

Example 5: Themed Diagrams

Using the blueprint theme:

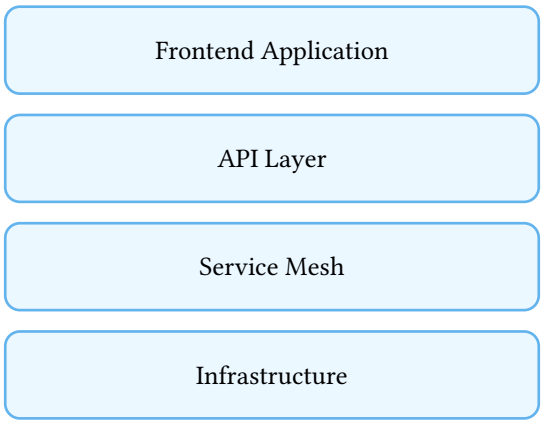

Figure 5: Blueprint theme with technical styling.

Using the warm theme:

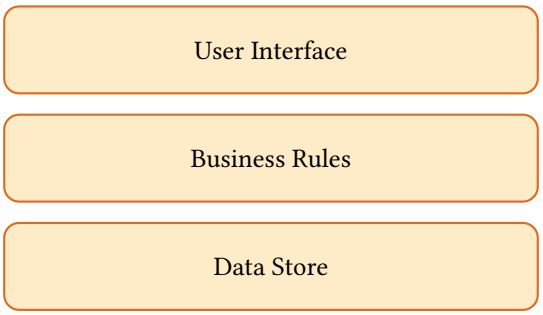

Figure 6: Warm theme with softer colors.

```
#themed-layers(theme-blueprint,
  [Frontend Application],
  [API Layer],
  [Service Mesh],
  [Infrastructure],
)

#themed-layers(theme-warm,
  [User Interface],
  [Business Rules],
  [Data Store],
)
```

Example 6: Full Architecture (4 Levels Deep)

A realistic microservices architecture diagram:

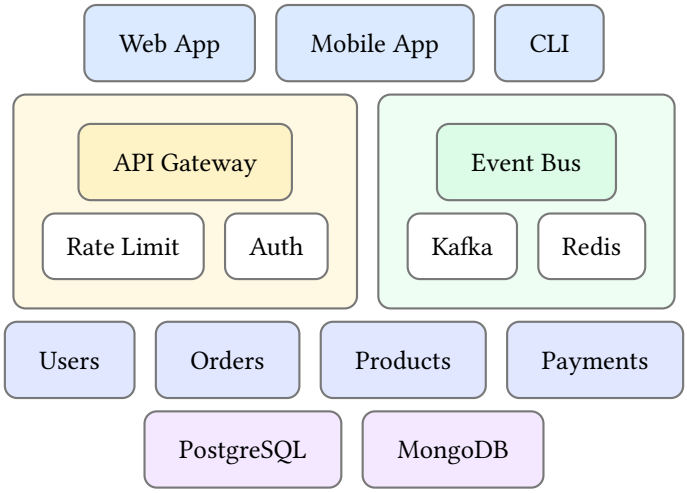

Figure 7: Microservices architecture with clients, gateway, services, and data stores.

Example 7: Figure with Caption

Using facade inside a figure:

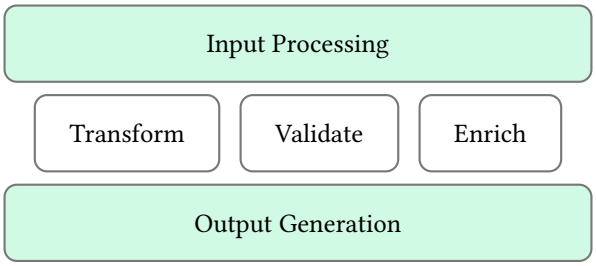

Figure 8: Data pipeline architecture showing the three-stage processing flow.

```
#figure(
  dg-layers(
    dg-rect([Input Processing], fill: rgb("#d1fae5"), width: 220pt),
    dg-flex(
```

```
      justify: "center",
      dg-rect([Transform], fill: white),
      dg-rect([Validate], fill: white),
      dg-rect([Enrich], fill: white),
    ),
    dg-rect([Output Generation], fill: rgb("#d1fae5"), width: 220pt),
  ),
  caption: [Data pipeline architecture.],
)
```

Function Reference

Shapes

`dg-rect(content, ..options)` — Rectangle block

- `width`, `height`: Dimensions (default: auto)
- `fill`: Background color
- `stroke`: Border (default: `0.8pt + luma(120)`)
- `radius`: Corner radius (default: `5pt`)
- `inset`: Padding (default: `(x: 8pt, y: 6pt)`)
- `content-align`: Content alignment (default: `center + horizon`)
- `header`: Header text/content displayed at top
- `header-fill`: Background color for header section
- `header-inset`: Padding for header (default: `(x: 8pt, y: 4pt)`)

`dg-circle(content, ..options)` — Circle block (same options, no radius)

`dg-ellipse(content, ..options)` — Ellipse block

Layouts

`dg-flex(direction, justify, align-items, gap, ..children)`

- `direction`: `"row"` or `"column"` (default: `"row"`)
- `justify`: `"start"`, `"center"`, `"end"`, `"space-between"`, `"space-around"`
- `align-items`: `"stretch"`, `"start"`, `"center"`, `"end"`
- `gap`: Spacing between items (default: `0.8em`)

`dg-layers(gap, align-items, ..children)`

- Vertical stack (shorthand for column flex)
- `gap`: Space between items (default: `0.5em`)
- `align-items`: Cross-axis alignment (default: `"center"`)

`dg-group(width, height, padding, fill, stroke, ..children)`

- Simple wrapper for grouping and adding space

Themes

Built-in themes: `theme-light`, `theme-dark`, `theme-blueprint`, `theme-warm`, `theme-minimal`

```
#themed-layers(theme-blueprint, [Layer 1], [Layer 2])
#themed-rect(theme-warm, [Block])
```

SysML Internal Block Diagram Style

A SysML-style IBD showing internal structure of a Vehicle Control System. This demonstrates the characteristic nested parts with typed instances, stereotypes, and port-like interfaces.

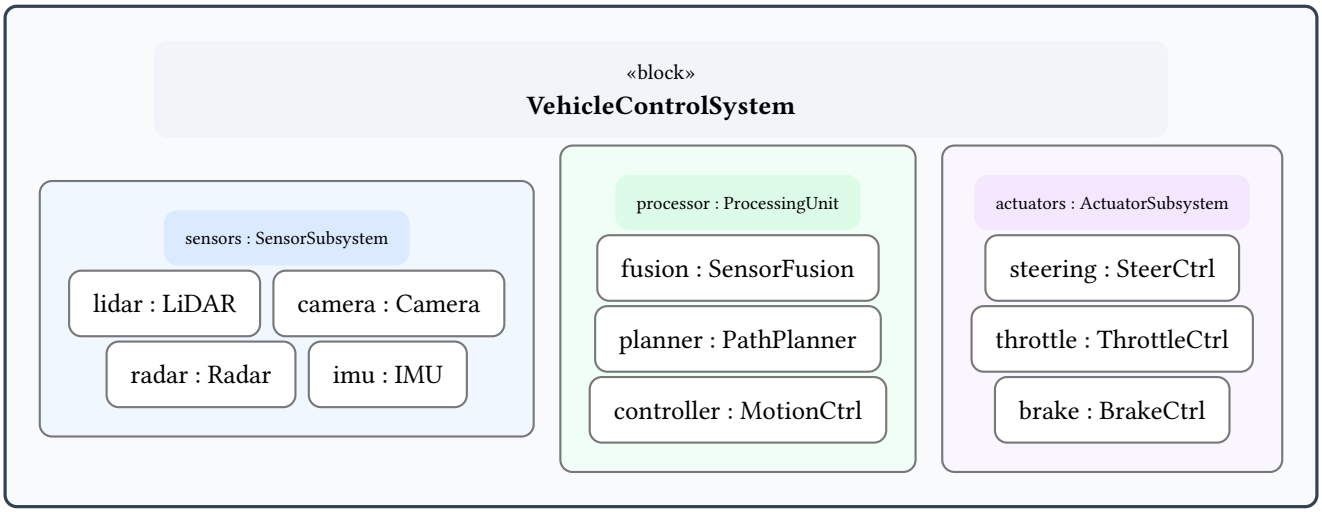

Figure 9: SysML Internal Block Diagram for a Vehicle Control System.

The diagram uses:

- **Stereotype headers** («`block`») identifying the block type
- **Typed parts** (`name : Type` notation) showing instances
- **Nested composition** showing internal structure of subsystems

```
dg-rect(
  dg-layers(
    // Stereotype header
    dg-rect([#text(size: 8pt)[«block»] \ #text(weight: "bold")[SystemName]], ...),
    // Internal parts row
    dg-flex(
      // Each part with typed name
      dg-rect(
        dg-layers(
          dg-rect([partName : PartType], ...),
          // Nested components...
        ),
        fill: color.lighten(60%),
      ),
    ),
  ),
  stroke: 1.2pt + rgb("#334155"),  // System boundary
)
```

AOSP Architecture Diagram

The standard Android Open Source Project architecture stack, showing the layered system design from Linux kernel to applications.

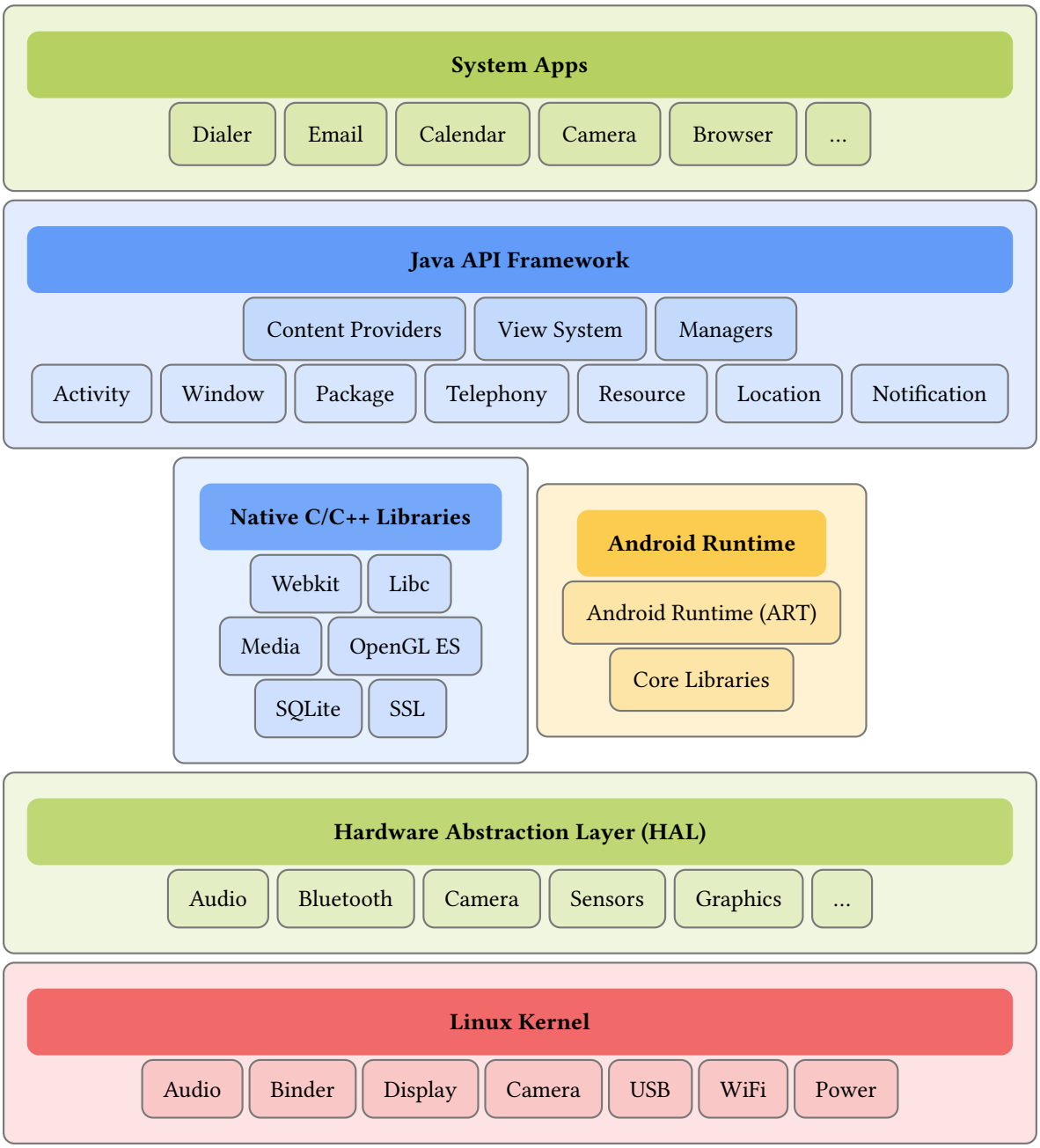

Figure 10: Android Open Source Project (AOSP) architecture stack.

The classic 5-layer Android stack:

- **System Apps** — Pre-installed applications (green)
- **Java API Framework** — Android SDK APIs and managers (blue)
- **Native Libraries + Runtime** — C/C++ libs and ART side-by-side (blue/yellow)
- **HAL** — Hardware abstraction interfaces (green)
- **Linux Kernel** — Drivers and core OS (red)

Decorators

Decorators add visual annotations to shapes like stereotypes, status indicators, and more.

Stereotype Labels

Add UML-style stereotype labels using the convenience `stereotype` parameter:

Figure 11: Stereotype labels for service, interface, and abstract types.

```
#dg-rect([UserService], stereotype: "service", fill: rgb("#dbeafe"))
#dg-rect([IRepository], stereotype: "interface", fill: rgb("#dcfce7"))
#dg-rect([BaseEntity], stereotype: "abstract", fill: rgb("#fef3c7"))
```

Status Indicators

Add status dots to show component health or state:

Figure 12: Status indicators showing healthy, warning, and error states.

```
#dg-rect([Database], status: green, fill: rgb("#f0fdf4"))
#dg-rect([Cache], status: yellow, fill: rgb("#fefce8"))
#dg-rect([External API], status: red, fill: rgb("#fef2f2"))
```

Combining Decorators

Use both stereotype and status together:

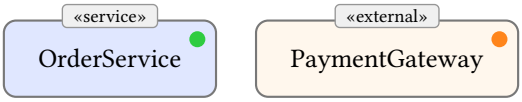

Figure 13: Combined stereotype labels and status indicators.

```
#dg-rect([OrderService], stereotype: "service", status: green)
#dg-rect([PaymentGateway], stereotype: "external", status: orange)
```

Explicit Decorator List

For full control, use the `decorators` parameter with constructor functions:

Figure 14: Explicit decorator list with custom positioning.

```
#dg-rect(
  [Component],
  decorators: (
    dg-stereotype("controller", fill: rgb("#dbeafe")),
    dg-status(blue, position: "top-right", size: 8pt),
  )
)
```

Stroke Presets

Use stroke presets for planned, optional, or tentative elements:

Figure 15: Stroke presets: solid, dashed, dotted, and planned styles.

```
#dg-rect([Current Feature], fill: rgb("#f0fdf4"))
#dg-rect([Planned Feature], stroke: stroke-dashed)
#dg-rect([Optional Module], stroke: stroke-dotted)
#dg-rect([Future Work], stroke: stroke-planned)
```

Container Headers

Headers provide title bars for container elements, useful for labeling nested diagrams and grouped components.

Basic Header

Add a header to any rectangle using the `header` parameter:

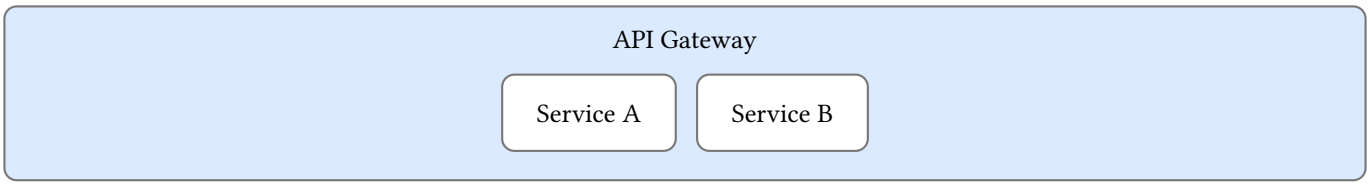

Figure 16: Simple header on a container element.

```
#dg-rect(
  dg-flex(
    dg-rect([Service A], fill: white),
    dg-rect([Service B], fill: white),
  ),
  header: "API Gateway",
  fill: rgb("#dbeafe"),
)
```

Header with Distinct Fill

Use `header-fill` to give the header a different background color:

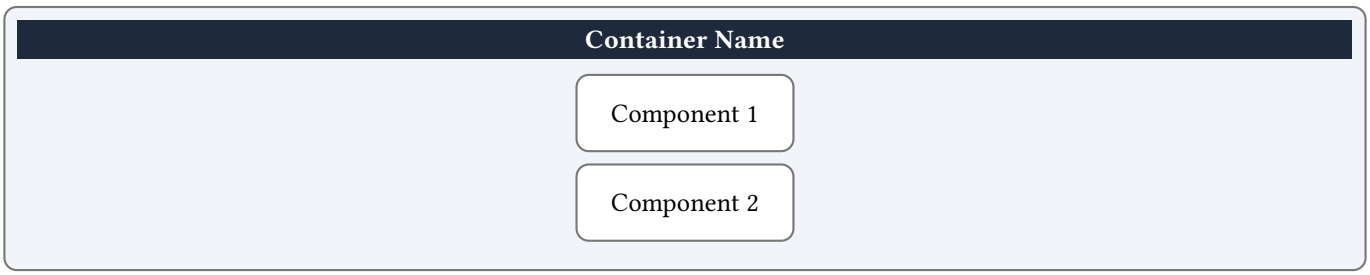

Figure 17: Header with distinct dark background.

```
#dg-rect(
  dg-layers(
    dg-rect([Component 1], fill: white),
    dg-rect([Component 2], fill: white),
  ),
  header: text(fill: white, weight: "bold")[Container Name],
  header-fill: rgb("#1e293b"),
  fill: rgb("#f1f5f9"),
)
```

Nested Containers with Headers

Headers work well with deeply nested structures:

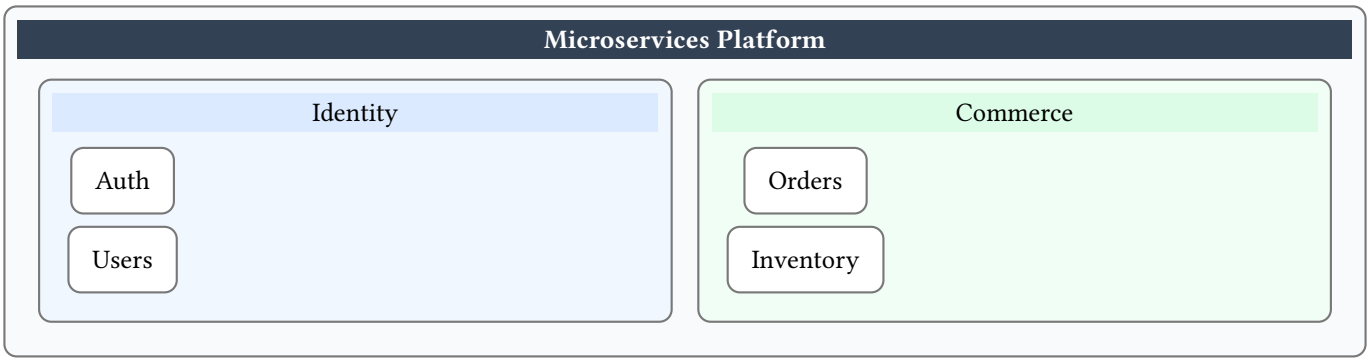

Figure 18: Nested containers with headers at multiple levels.

```
#dg-rect(
  dg-flex(
    gap: 1em,
    dg-rect(
      dg-layers(..),
      header: "Identity",
      header-fill: rgb("#dbeafe"),
      fill: rgb("#dbeafe").lighten(60%),
    ),
    dg-rect(
      dg-layers(..),
      header: "Commerce",
      header-fill: rgb("#dcfce7"),
      fill: rgb("#dcfce7").lighten(60%),
    ),
  ),
  header: text(fill: white, weight: "bold")[Microservices Platform],
  header-fill: rgb("#334155"),
  fill: rgb("#f8fafc"),
)
```

Headers with Decorators

Headers can be combined with stereotype and status decorators:

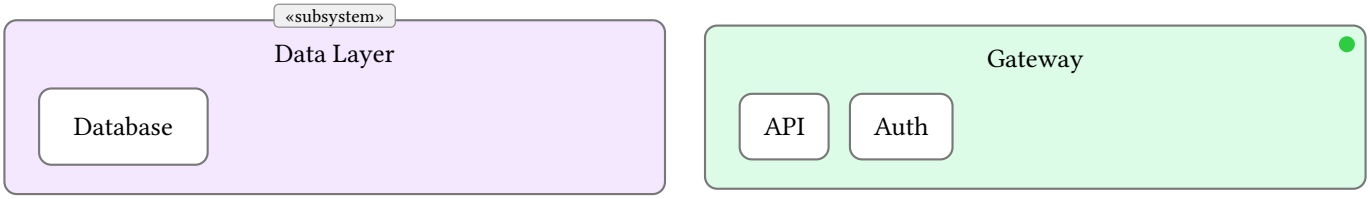

Figure 19: Headers combined with stereotype labels and status indicators.

```
#dg-rect(
  dg-rect([Database], fill: white),
  header: "Data Layer",
  fill: rgb("#f3e8ff"),
  stereotype: "subsystem",
)
```

Landscape Full-Page Diagram

For large diagrams, use a landscape page. Wrap your diagram in a `page()` call with `flipped: true`:

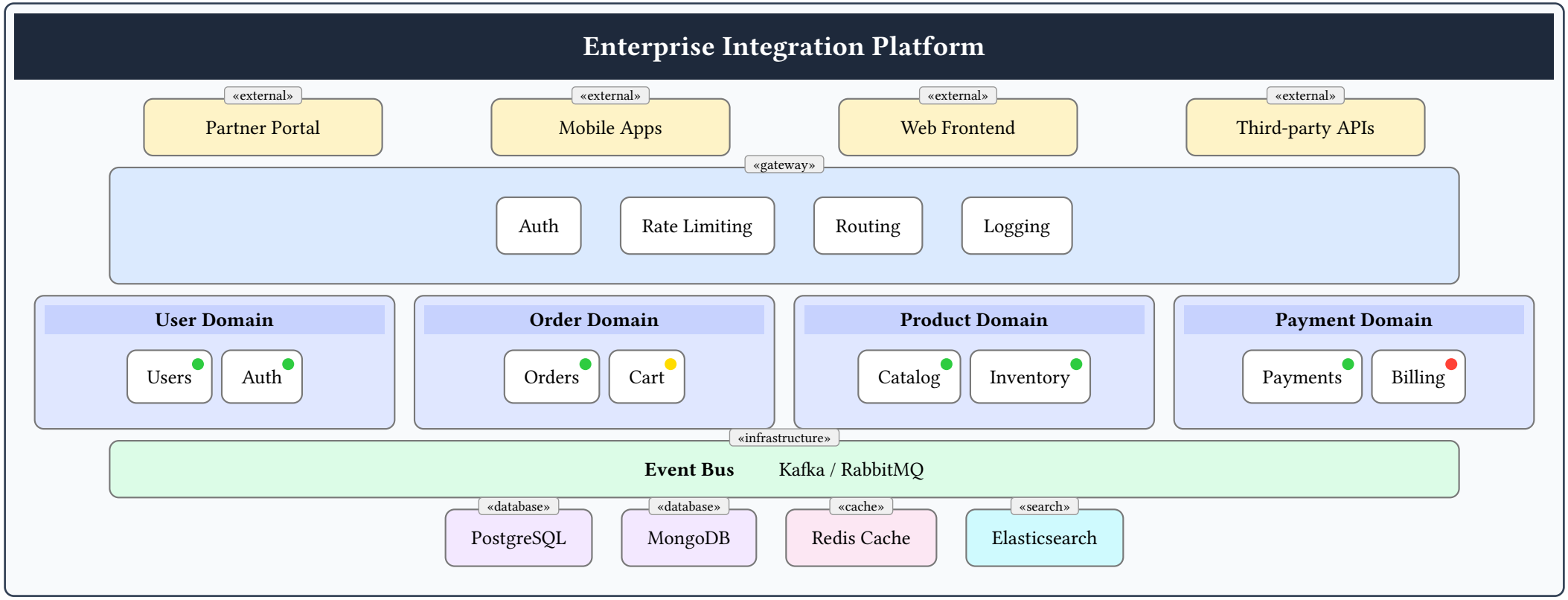

Figure 20: Enterprise Integration Platform — full landscape page layout with domain-driven microservices.

```
#page(flipped: true, margin: 1cm)[
  #dg-rect(
    dg-layers(
      // External systems row
      dg-flex(...),
      // Domain services with headers
      dg-flex(
        dg-rect(
          dg-flex(...services...),
          header: text(weight: "bold")[User Domain],
          header-fill: rgb("#c7d2fe"),
          fill: rgb("#e0e7ff"),
        ),
        // More domains...
      ),
      // Data layer
      dg-flex(...),
    ),
    header: text(fill: white, weight: "bold")[Platform Name],
    header-fill: rgb("#1e293b"),
    fill: rgb("#f8fafc"),
  )
]
```

Deep Recursion Example

Here's a multi-level architecture diagram showing nested components:

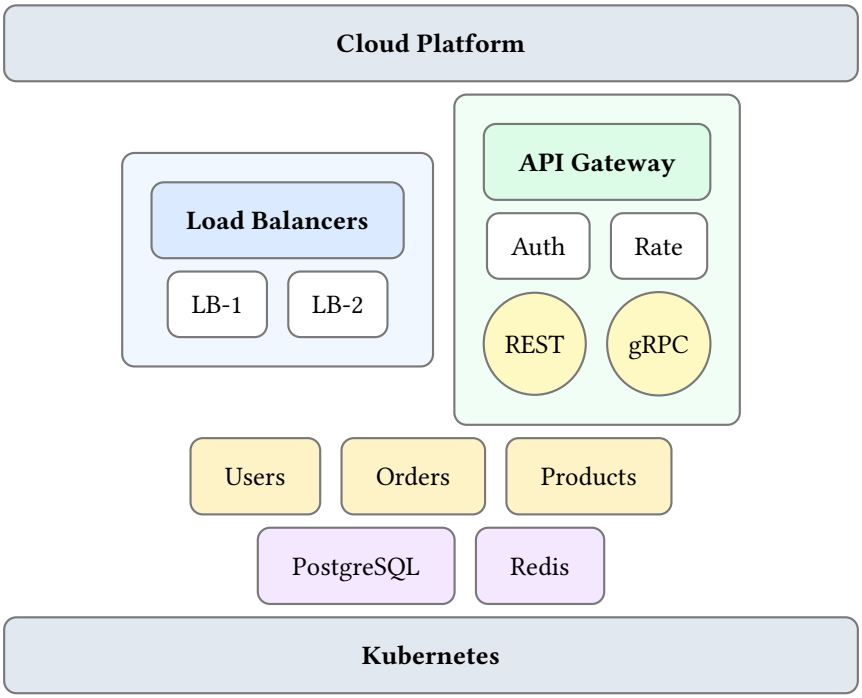

Figure 21: Cloud platform architecture with deeply nested components.