Pipeline demo: what the visitor should experience

Twelve stories for a demo that a curious person launches with one command. Three services that know nothing about each other stand behind one router, and two separate web apps in front of it show the same data changing. The moment that matters is a bottleneck moving.

The visitor

A developer or architect at a small engineering firm, evaluating whether federation could connect the tools they already have. Has Docker, a browser and fifteen minutes.

The model owner

A systems engineer who writes the SysML v2 model and wants it to stay the source of truth.

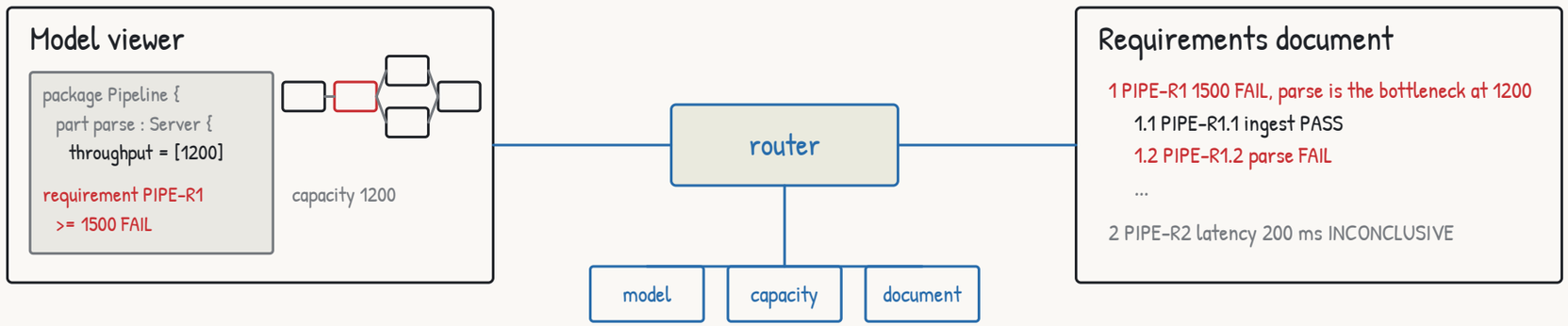
The document owner

A requirements engineer who owns ordering, numbering and inclusion in the document, and has never seen SysML.

The journey, in order

01 Launch with one command	02 Read the model: text and wiring	03 Raise a server that is not the bottleneck: nothing moves	04 Raise the bottleneck: it migrates, then the requirement passes	05 Tighten the limit: red, with the servers responsible named	06 Read the document: own numbering, model relationships, verdicts
07 Reorder and nest, the model untouched	08 Shape the document: headings, prose, exclude and restore	09 Change a value from the document, the viewer follows	10 Change from the viewer, watch the document	11 Query the graph: three services in one answer	12 Reset to the shipped state

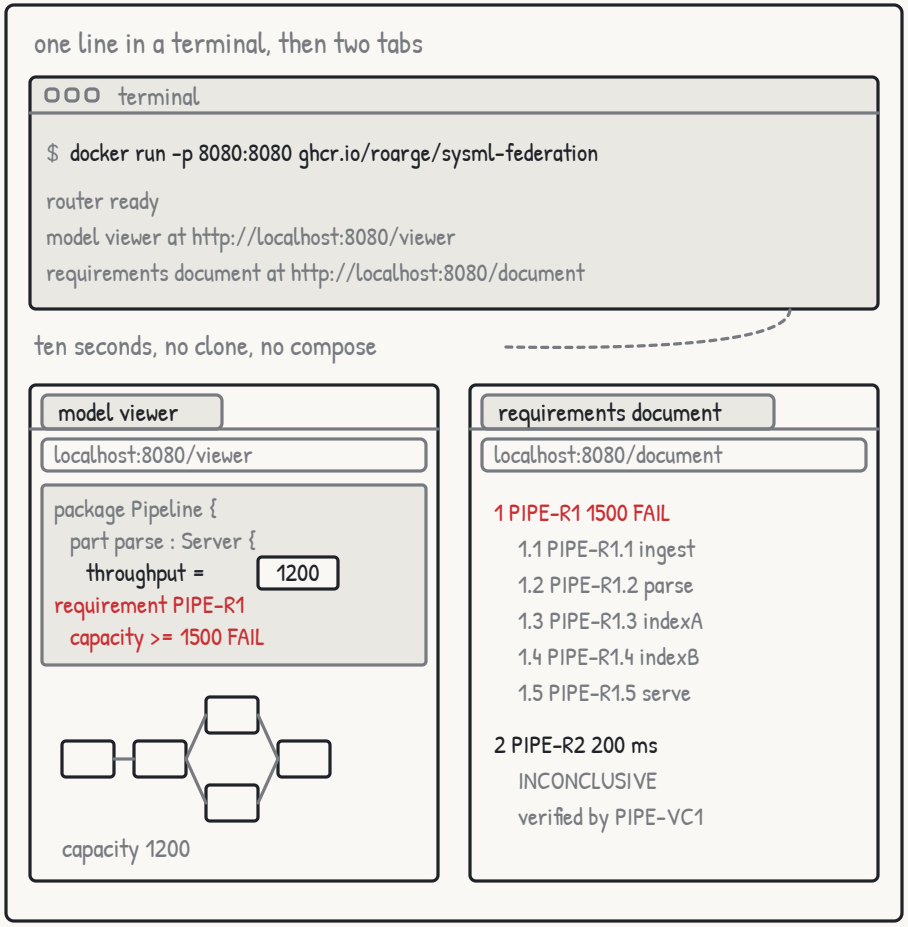
two apps, one graph, three services that never meet



visitor story · model owner story · document owner story · a failing requirement or the bottleneck, the only red on any board · shared through the router

US-01 Launch with one command

the visitor



Sketch of the terminal after the command, with the two tabs it names in their shipped state.

As **the visitor**, I want to start the whole demo with one command copied from the README, so that I can see it working before deciding whether to read on.

Acceptance

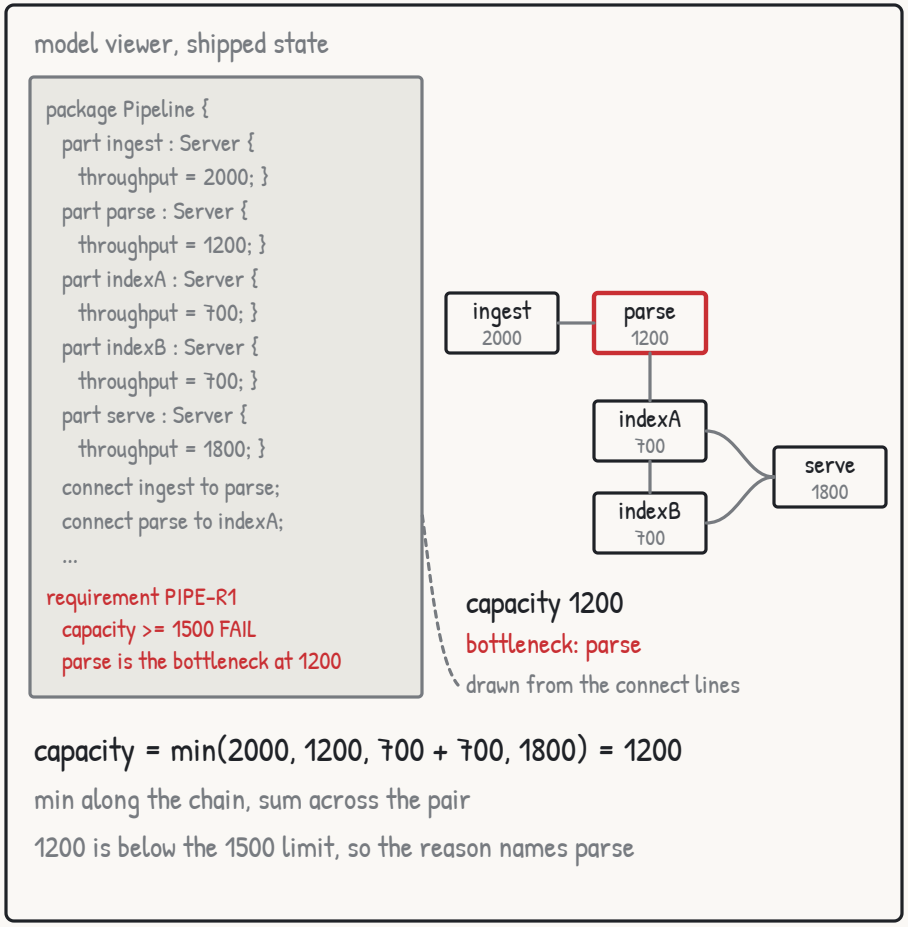
1. **Given** the image is already pulled, **when** the one docker run line from the README is executed, **then** within ten seconds both addresses named in the README render the model viewer and the requirements document when opened.
2. **Given** the container is running with no route to the internet, **when** either app is opened, **then** it renders fully and every request it makes goes to the one published port.

Why it matters

The README makes an argument and this is where the visitor stops reading and starts looking. If the launch costs more than one line, the bottleneck moving is never seen.

US-02 Read the model

the visitor



Sketch of the viewer as it opens, with nothing edited yet.

As **the visitor**, I want to see the pipeline model as SysML v2 text beside a sketch of the wiring, so that I understand what is being modelled without knowing the language.

Acceptance

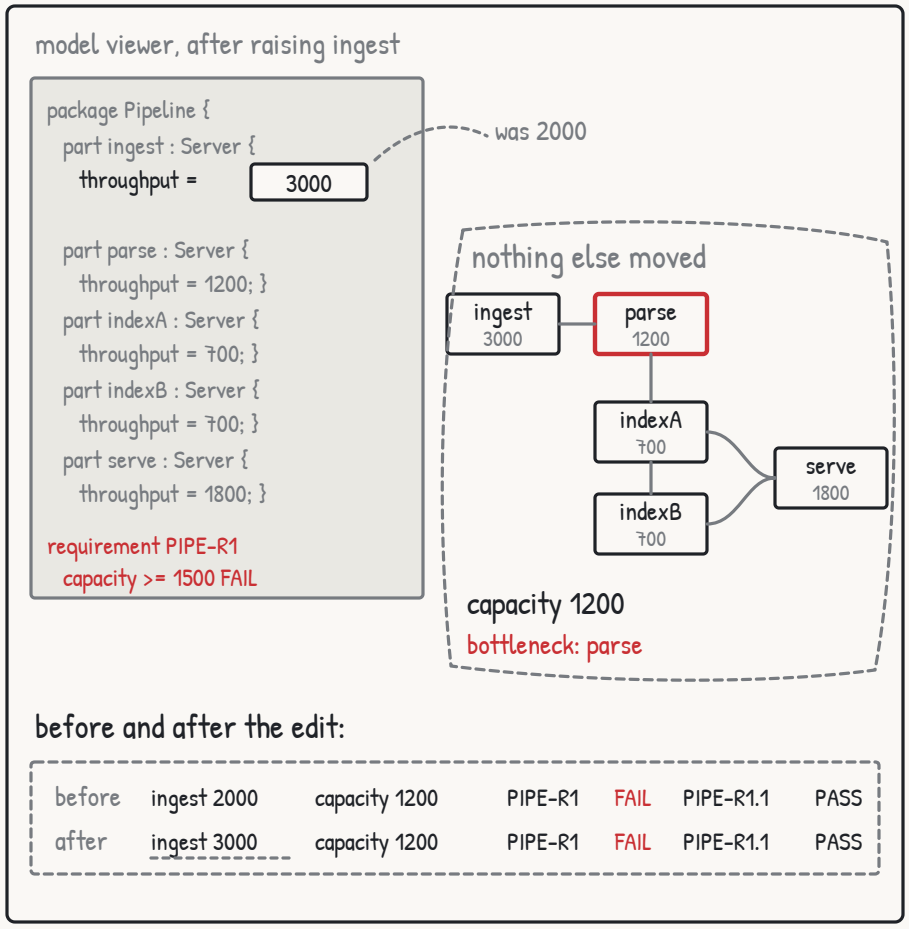
1. **Given** the shipped model, **when** the viewer opens, **then** the text pane shows the model file with the servers, their throughput values, the connections, the requirements and their short names.
2. **Given** the shipped model, **when** the viewer opens, **then** the sketch shows the five servers left to right as wired, each with its throughput, the rolled-up capacity of 1200, and parse marked as the bottleneck.
3. **Given** the shipped model, **when** the viewer opens, **then** PIPE-R1 shows its limit of 1500, a verdict of FAIL and a reason naming parse at 1200.

Why it matters

The viewer puts the model text and the wiring side by side, so a visitor who has never read SysML can see what the five servers are and why the capacity is 1200. Every later story edits something on this screen, and this is the state those edits start from.

US-03 Raise a server that is not the bottleneck

the visitor



Sketch of the viewer after ingest is raised, with the shipped and edited states compared below.

As **the visitor**, when I raise the throughput of a server that is not the bottleneck, I want to see that the capacity, the verdict and the bottleneck stay as they were, so that I learn that a chain is governed by its worst link.

Acceptance

1. **Given** the shipped model, **when** ingest is raised from 2000 to 3000, **then** the capacity stays 1200, PIPE-R1 stays FAIL, the bottleneck stays parse, and the only other visible change is PIPE-R1.1 continuing to pass with its new value.
2. **Given** the shipped model, **when** a throughput is set to a value that is not a number or is negative, **then** the value is rejected and the previous value stands.
3. **Given** the shipped model, **when** parse is set to 0, **then** the capacity becomes 0, PIPE-R1 stays FAIL and the reason names parse at 0.

What does not happen

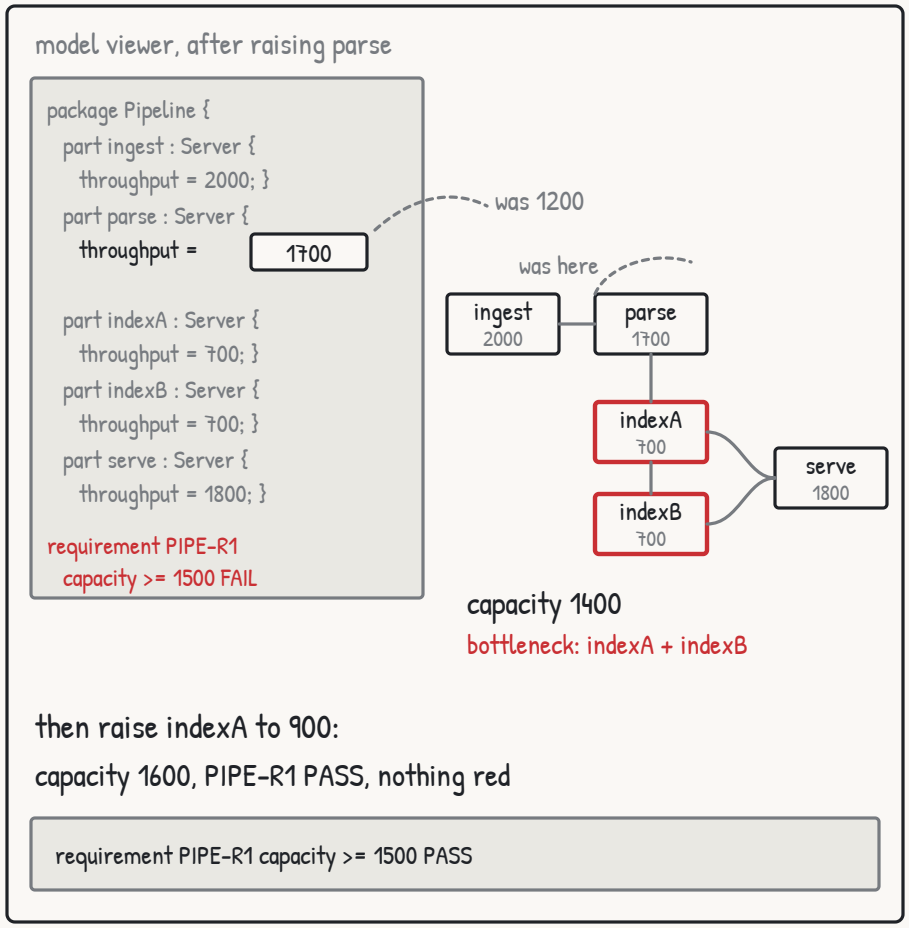
No verdict flips, no colour changes, the sketch does not move its bottleneck mark. The story uses ingest because raising indexA or indexB would leave the capacity alone but could flip that server's own derived requirement, which is US-10's lesson rather than this one.

Why it matters

A change that goes nowhere shows the visitor that capacity belongs to the whole chain. It is the quiet half of the contrast that US-04 completes.

US-04 Raise the bottleneck

the visitor



Sketch of the viewer after the first edit, with the second edit noted below it.

As **the visitor**, when I raise the throughput of the bottleneck, I want to see the capacity rise and the bottleneck move, so that I see the model respond as a whole.

Acceptance

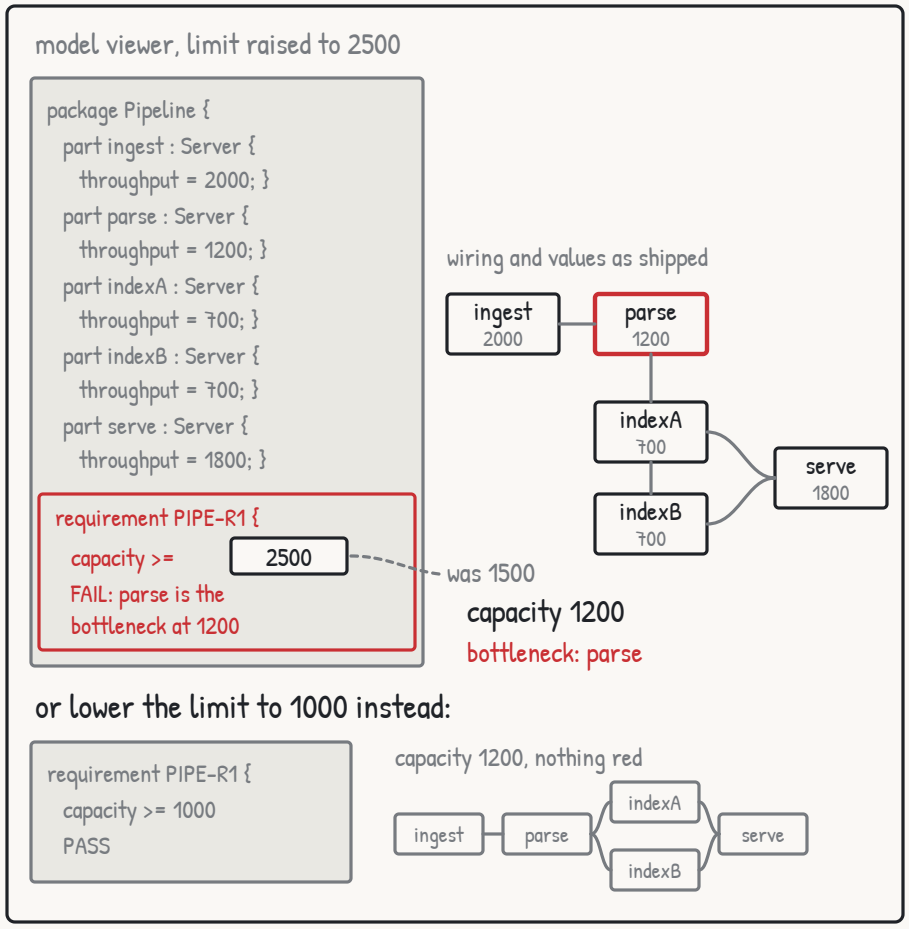
1. **Given** the shipped model, **when** parse is raised from 1200 to 1700, **then** the capacity becomes 1400, PIPE-R1 stays FAIL, and the bottleneck moves to the pair indexA and indexB.
2. **Given** parse at 1700, **when** indexA is raised from 700 to 900, **then** the capacity becomes 1600, PIPE-R1 becomes PASS with a reason naming the index pair at 1600 against a limit of 1500, and the requirement block is no longer red.

Why it matters

The rollup is min over the chain and sum over the parallel pair. Raising the bottleneck exposes the next cut, which is what makes the effect surprising before it is seen and obvious after.

US-05 Tighten the limit

the model owner



Sketch of the viewer after the limit is raised to 2500, with the limit at 1000 noted below in pencil.

As **the model owner**, I want to change the limit in the throughput requirement and see which servers are responsible when it fails, so that the requirement is checked against the model rather than against a copy.

Acceptance

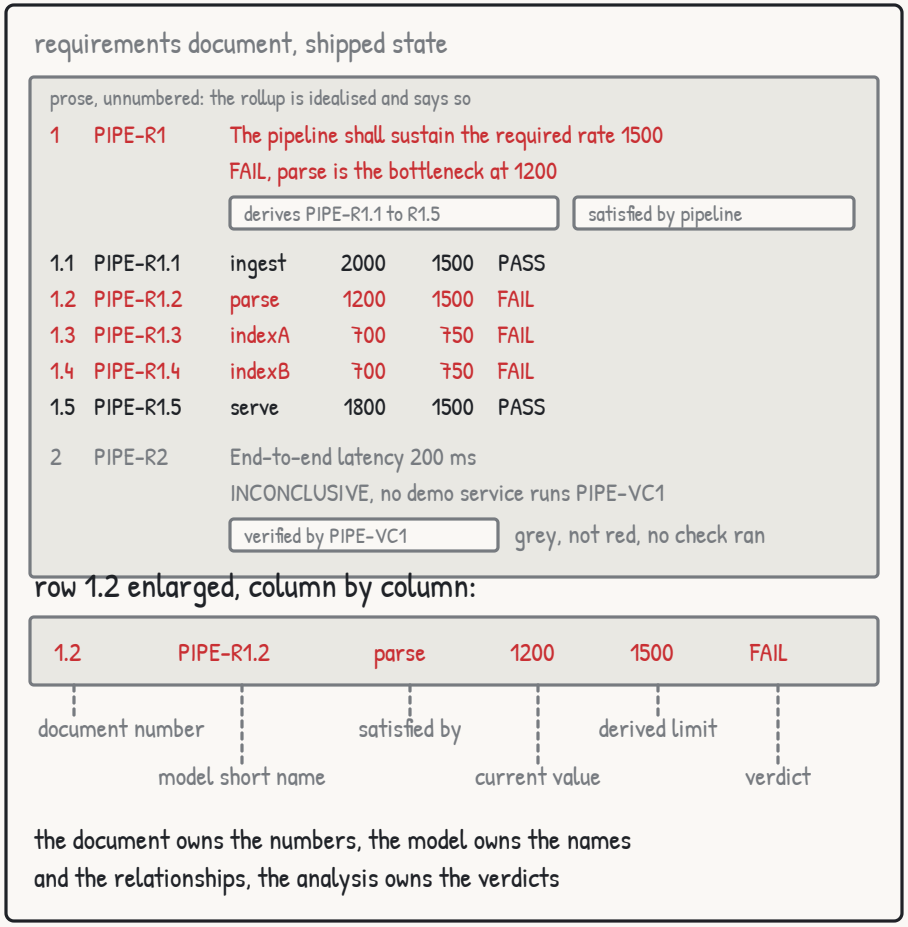
1. **Given** the shipped model, **when** the limit of PIPE-R1 is changed from 1500 to 1000, **then** the verdict becomes PASS.
2. **Given** the shipped model, **when** the limit of PIPE-R1 is changed from 1500 to 2500, **then** the verdict becomes FAIL, the requirement block turns red, and the reason names parse as the bottleneck at 1200.
3. **Given** any value edited in the viewer, **when** the model text is read, **then** the edited number appears in the text exactly where the original literal was.

Why it matters

The limit is a literal in the model text and the verdict comes from a service that has never parsed that text, so moving one and watching the other answer shows the check running against the model itself. The third criterion is the model owner's assurance that the edit landed in the source and not in a copy.

US-06 Read the document

the document owner



Sketch of the document in the shipped state, with row 1.2 enlarged below it.

As **the document owner**, I want to read the requirements as a numbered document that also shows what the model knows about each one, so that I can review them without opening a modelling tool.

Acceptance

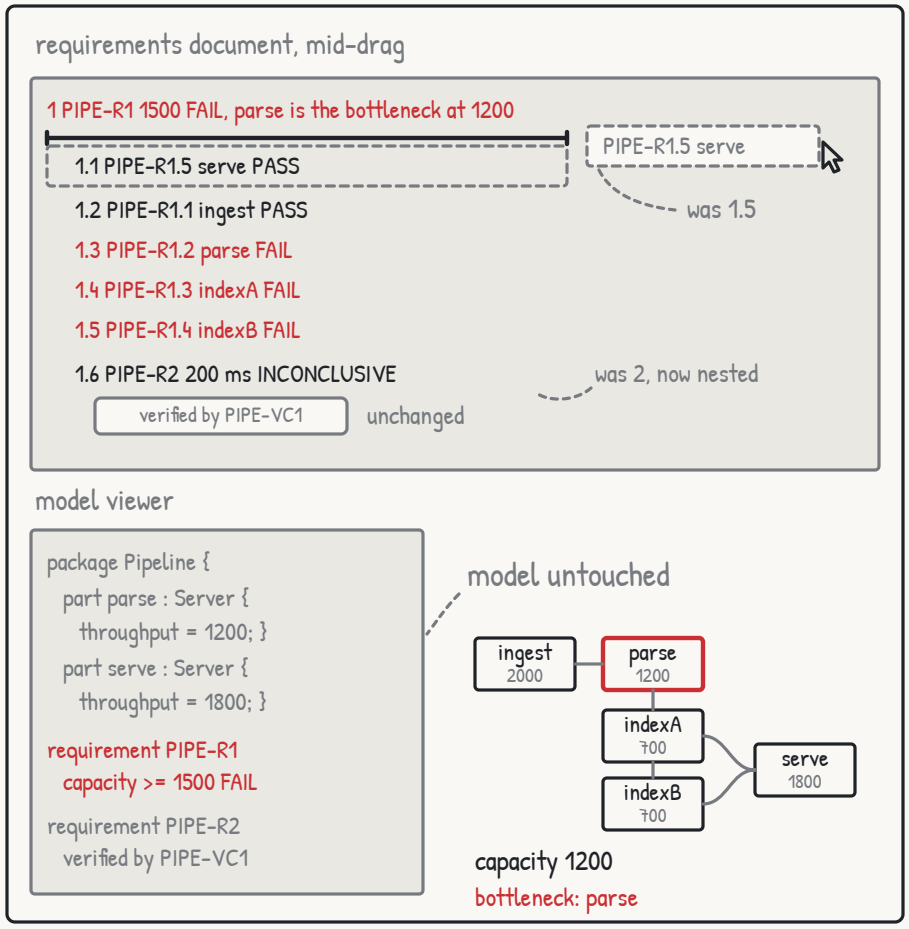
1. **Given** the shipped model and the shipped document structure, **when** the document opens, **then** it shows PIPE-R1 as section 1 with its derived requirements as 1.1 to 1.5 and PIPE-R2 as section 2, numbers that are the document's own and not the model's short names.
2. **Given** the document is open, **when** a requirement is read, **then** it shows its short name, text, limit, the requirement it is derived from or the requirements derived from it, the part that satisfies it, the verification case that verifies it, the current value of the subject it is checked against, and its verdict with a reason.
3. **Given** the document is open, **when** PIPE-R2 is read, **then** its verdict is INCONCLUSIVE with the reason "PIPE-VC1 is declared and no service runs it".

Why it matters

The document owner reads relationships from the model and verdicts from the analysis without opening a modelling tool, in a document whose numbering is its own. The service behind this document has never parsed a model file, which is the federation argument on one screen.

US-07 Reorder and nest

the document owner



Sketch of the document mid-drag, numbering already recomputed, PIPE-R2 nested under PIPE-R1 and the viewer unchanged below.

As **the document owner**, I want to move requirements into a different order and nest one under another, so that the document reads the way my readers expect while the model stays untouched.

Acceptance

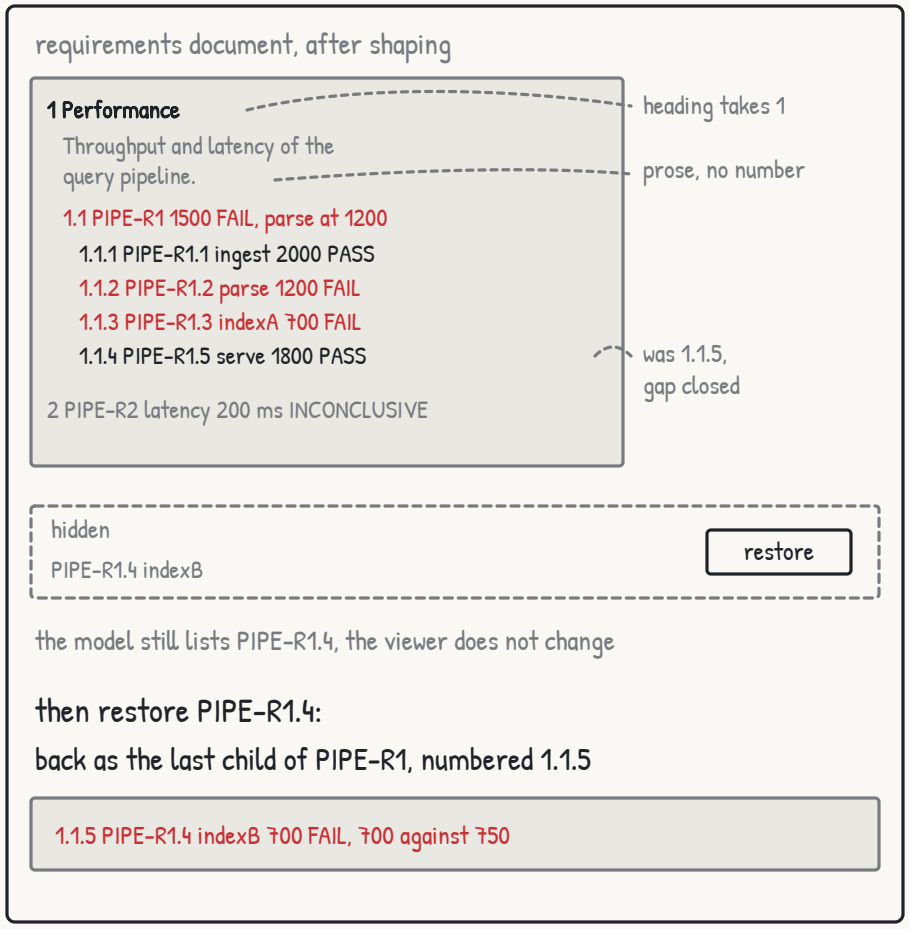
1. **Given** the shipped document, **when** PIPE-R1.5 is moved above PIPE-R1.1, **then** PIPE-R1.5 becomes 1.1, the others shift to 1.2 to 1.5 in their former order, and nothing else changes number.
2. **Given** the shipped document, **when** PIPE-R2 is moved under PIPE-R1, **then** it is numbered 1.6, and its relationships in the model are unchanged and still shown.
3. **Given** a reordered document, **when** the model viewer is read, **then** the model text and the sketch are unchanged.

Why it matters

Ordering and nesting are editorial decisions held by the document service alone, so the document owner can shape the reading order without touching a model they have never seen. The relationships and verdicts still arrive from the other two services, which shows three services holding different facts about one requirement.

US-08 Shape the document

the document owner



Sketch of the document with the heading, the prose and the exclusion in place, with the restore noted below it.

As **the document owner**, I want to add headings and prose and hide a requirement, so that the document carries the editorial decisions the model does not contain.

Acceptance

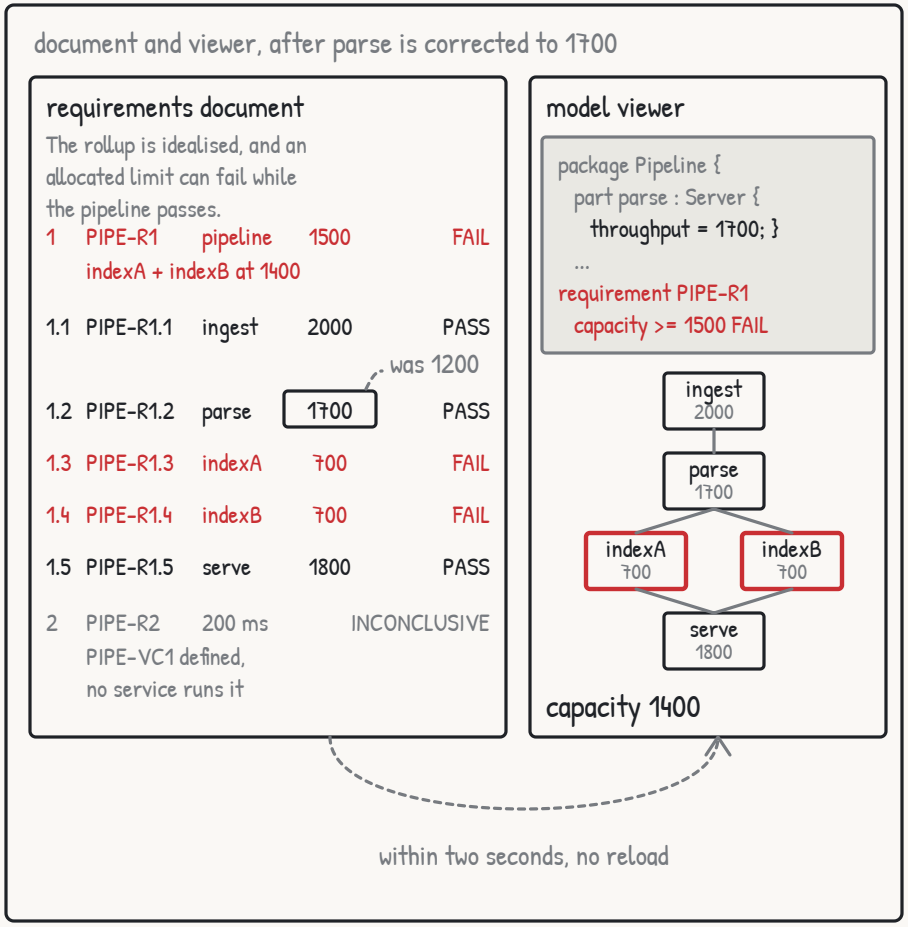
1. **Given** the shipped document, **when** a heading "Performance" is inserted as the parent of PIPE-R1, **then** the heading takes number 1, PIPE-R1 becomes 1.1 and its derived requirements 1.1.1 to 1.1.5.
2. **Given** the document is open, **when** a prose paragraph is added under the heading, **then** it appears in place and carries no number.
3. **Given** the shipped document, **when** PIPE-R1.4 is excluded, **then** it disappears from the document, PIPE-R1.5 becomes 1.4, and the model still lists PIPE-R1.4.
4. **Given** PIPE-R1.4 is excluded, **when** it is restored, **then** it returns as the last child of PIPE-R1 and is numbered 1.5.

Why it matters

A heading, a paragraph and an exclusion are editorial decisions with no home in the model, and the document service keeps them while the model stays as it was. That is the README's claim at small scale, a service that has never parsed a model file attaching its own data to the model's objects.

US-09 Change a value from the document

the document owner



Sketch of the document after parse is corrected to 1700, with the viewer beside it already following.

As **the document owner**, I want to correct a number inline in the document, so that the model is updated from where I work.

Acceptance

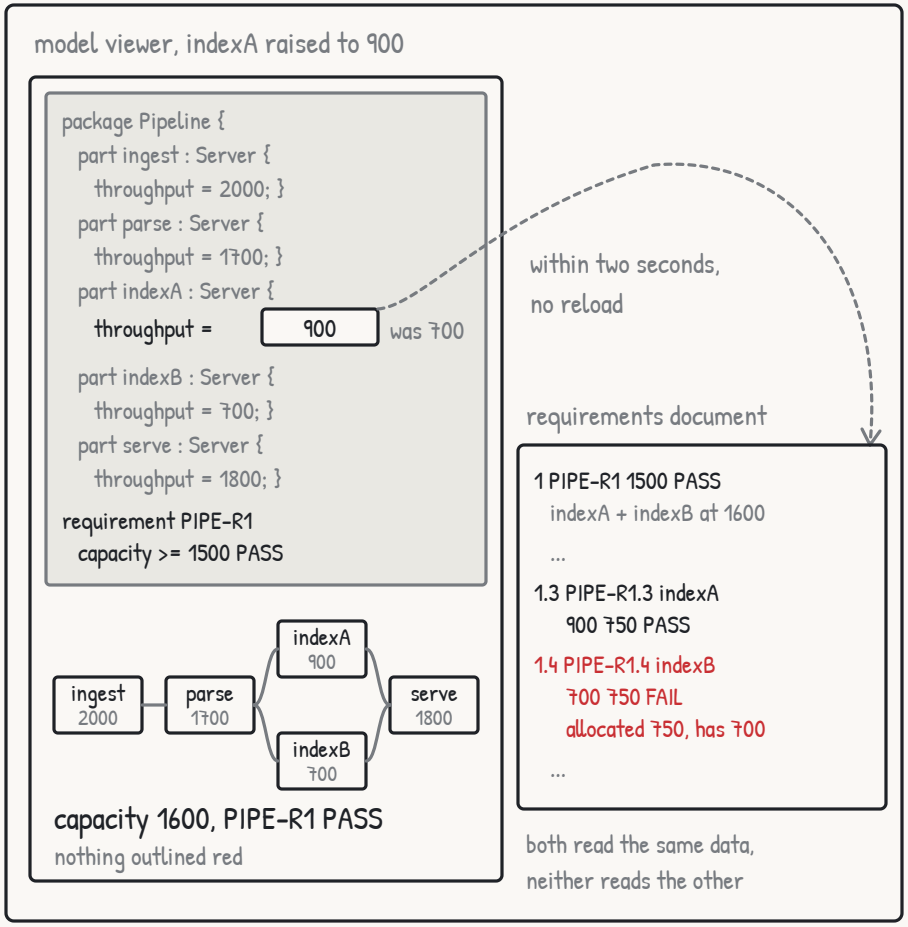
1. **Given** the viewer is open in another tab, **when** the throughput of parse is changed from 1200 to 1700 in the row of PIPE-R1.2, **then** PIPE-R1.2 becomes PASS, PIPE-R1 stays FAIL with a reason now naming indexA and indexB at 1400, and within two seconds the viewer's text shows 1700, its sketch shows capacity 1400 with the index pair marked, all without a reload.
2. **Given** the document is open, **when** the limit of PIPE-R1 is changed in its row, **then** the viewer shows the new limit in the requirement text.

Why it matters

The document owner has never seen SysML and still corrects the model from the page they already work in. The viewer following within two seconds shows that the edit landed in the served model rather than in a copy held by the document service.

US-10 Change from the viewer, watch the document

the visitor



Both apps after indexA is raised to 900 in the viewer, the document beside it already showing the new verdicts.

As **the visitor**, when I edit a value in the viewer with the document open beside it, I want the document to update by itself, so that I see that both apps read the same data rather than each other.

Acceptance

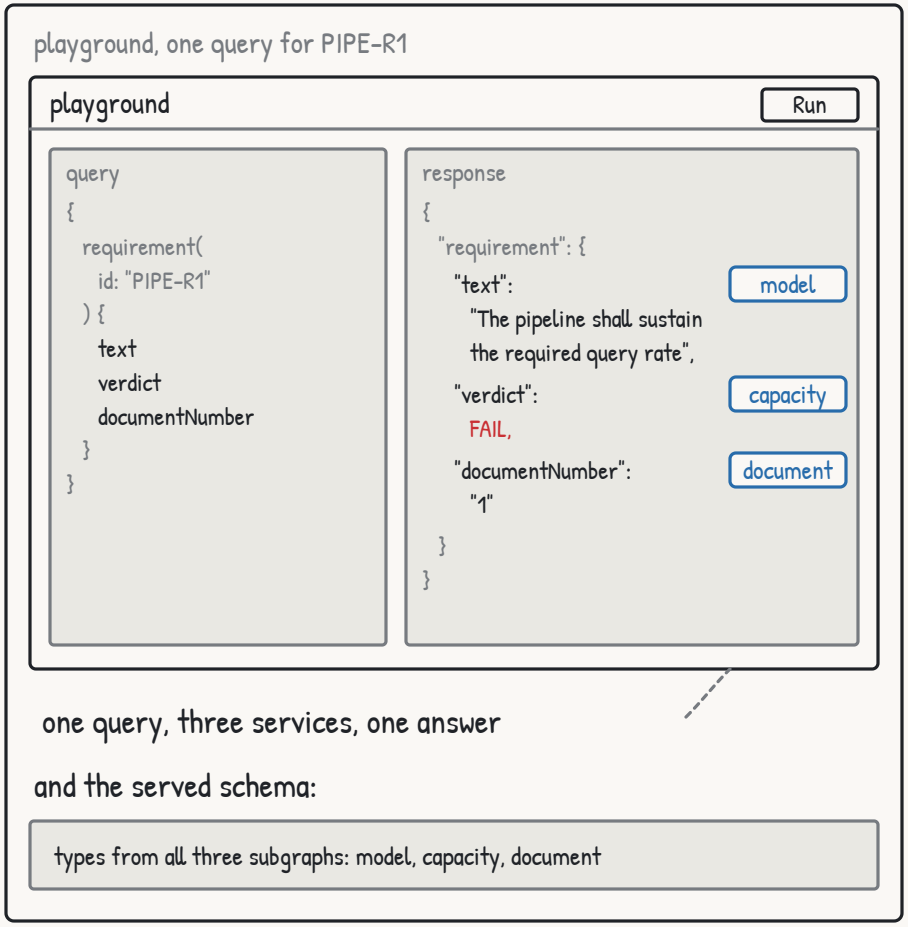
1. **Given** both apps are open and parse is at 1700, **when** indexA is raised from 700 to 900 in the viewer, **then** within two seconds and without a reload the document shows PIPE-R1 as PASS with a reason naming the index pair at 1600 against 1500, PIPE-R1.3 as PASS, and PIPE-R1.4 still FAIL with a reason naming indexB at 700 against its allocated 750.

Why it matters

The document service has never parsed a model file, yet its verdicts change the moment the model does. Watching that happen in a second window, unprompted, is the visitor's evidence that both apps read one graph rather than each other.

US-11 Query the graph

the visitor



Sketch of the playground after one query, each response field tagged with the service that answered it.

As **the visitor**, I want to run one query that returns fields from all three services for one requirement, so that I see the federation rather than take it on trust.

Acceptance

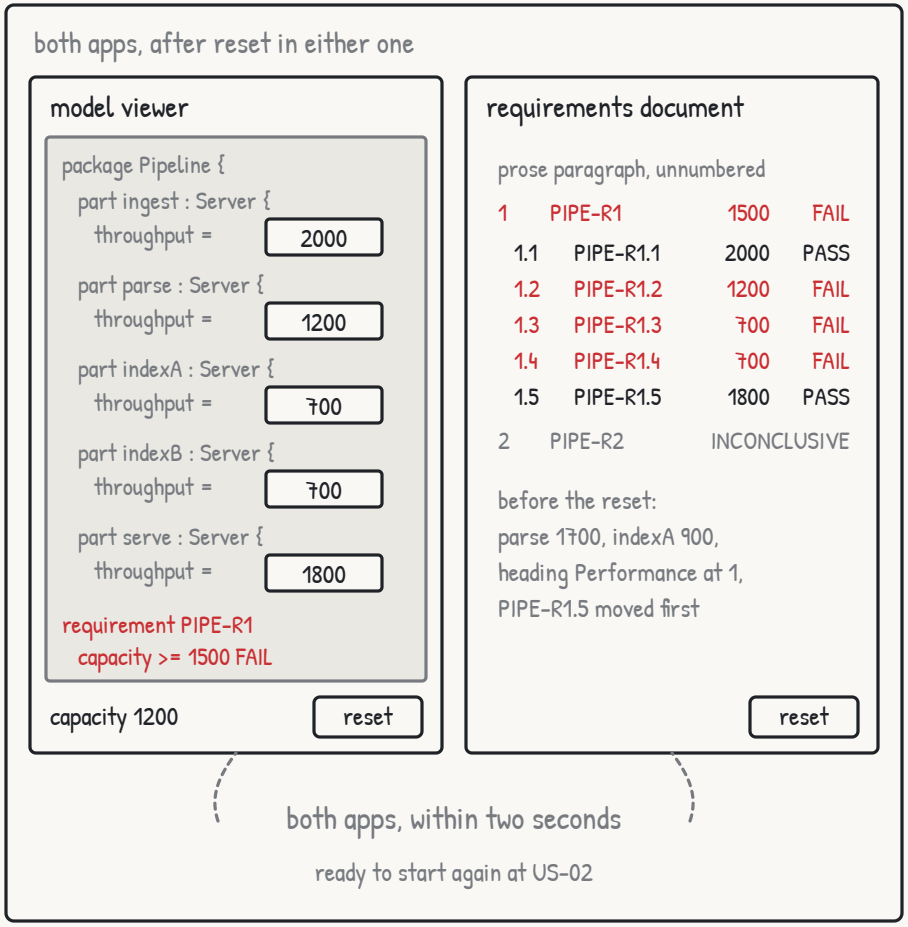
1. **Given** the demo is running, **when** one query in the playground asks for PIPE-R1's text, verdict and document number, **then** one response carries all three, and the served schema contains types contributed by all three subgraphs.

Why it matters

The two apps show the data being shared, but a visitor could still suspect that one app calls the other. One response with a field from each service settles it, because the services never meet and the router joins them on the key PIPE-R1.

US-12 Reset

the visitor



Sketch of both apps just after a reset, every value and number back to the shipped state.

As **the visitor**, I want to return the demo to its shipped state, so that I can show it again from the beginning.

Acceptance

1. **Given** values and the document structure have been edited, **when** the reset control is used in either app, **then** both apps show the shipped values and the shipped document structure within two seconds.

Why it matters

Without it the demo runs once per container start. One control in either app restoring both is the shared state of US-09 and US-10, shown a final time.