COMMUNITY
THE ASF CONFERENCE
CODE

Lessons Learned: Building Service Gateway with Tomcat And Virtual Threads

Yike Xiao | 智联招聘

About Me

- 汪苏诚 智联招聘
- 萧易客是网名
- **Working on frameworks and infrastructure**
- github.com/shawyeok
- Apache Pulsar/Bookkeeper/Zookeeper Contributor

COMMUNITY
THE ASF CONFERENCE
CODE

Disclaimer

- This talk is based on real project experience.
- All opinions are my own and do not represent official positions.
- The goal is to share lessons, not promote specific tools.

COMMUNITY
THE ASF CONFERENCE
CODE

CONTENTS

1. Why We Built Our Own Gateway
2. Design Choices
3. Engineering Considerations
4. Migration & Rollout
5. Challenges & Takeaways

COMMUNITY
THE ASF CONFERENCE
CODE

PART 01

Why We Built Our Own Gateway

Why Build a New Service Gateway?

1. **High Maintenance Cost**
   - **Steep learning curve for new developers**
   - **Adding new APIs requires modifying configuration files or code**
2. **Low Iteration Efficiency**
   - API changes need **full redeployment** of gateway services
   - Existing gateway runs **~500 instances**
   - A single rolling deployment cycle takes **20–30 minutes**

Goal: Simplify integration and improve iteration speed while keeping operational costs low.

Why Not Use Existing Open-Source Gateways?

Kong Gateway, Apache APISIX, Apache Shenyu, etc

- Avoid introducing a new tech stack that increases learning costs
- Need tight integration with internal tools and infrastructure
- Plugin-based configuration is **not friendly** for our business developers
- Route configuration sync is complicated across multiple environments
- Open-source solutions require **hacks** for smooth migration

COMMUNITY
THE ASF CONFERENCE
CODE

PART 02

Design Choices

Spring Cloud Gateway

- **Spring Cloud Gateway Server WebFlux**
  - **Based on Spring Boot, WebFlux, and Project Reactor**
- **Spring Cloud Gateway Server Web MVC**
  - **Based on Spring Boot, Spring WebMvc.fn**

Spring WebMvc.fn

Functional Programming Model

- HandlerFunction
- RouterFunction
- HandlerFilterFunction

Deterministic Execution

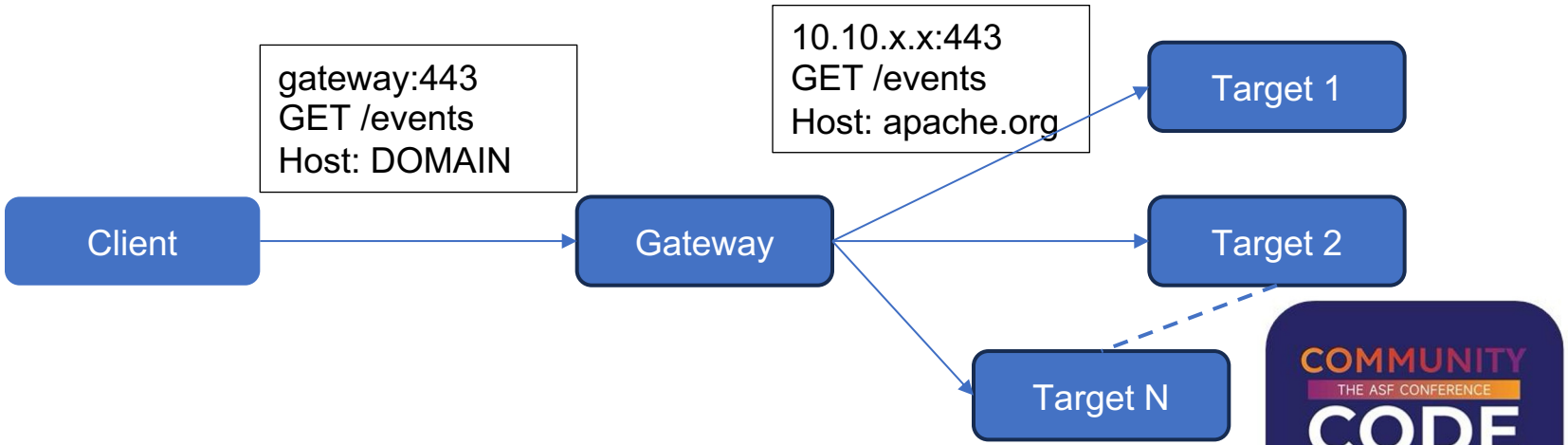
Easier to compose request processing pipeline

Composing a Request Processing Pipeline

http(http://apache.org) // HandleFunction

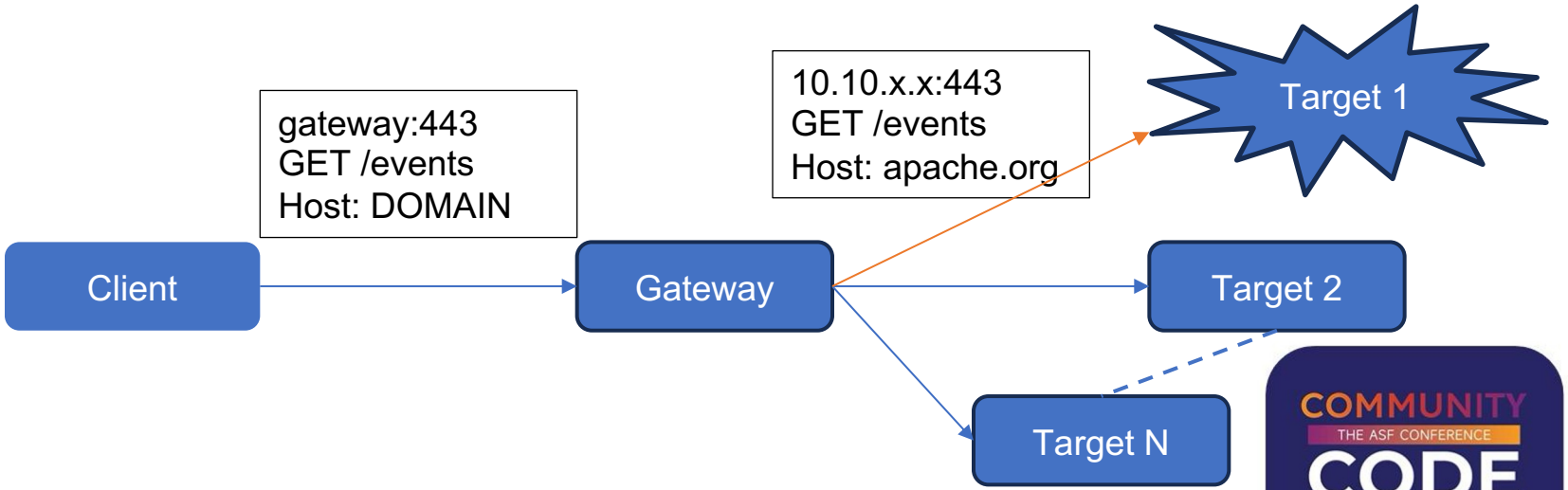
Composing a Request Processing Pipeline

```
lb( “jobservice” )                                  // HandlerFilterFunction
  .apply( http() )                                  // HandlerFunction
```



Composing a Request Processing Pipeline

```
retry(3)                                          // HandlerFilterFunction
  .andThen( lb( "jobservice" ) )                  // HandlerFilterFunction
  .apply( http() )                                // HandlerFunction
```



Composing a Request Processing Pipeline

```
tracing()                                          // HandlerFilterFunction
  .andThen( metrics() )                            // HandlerFilterFunction
  .andThen( logging() )                            // HandlerFilterFunction
  .andThen( retry(3) )                             // HandlerFilterFunction
  .andThen( lb( “jobservice” ) )                   // HandlerFilterFunction
  .apply( http() )                          // HandlerFunction
```

Composing a Request Processing Pipeline

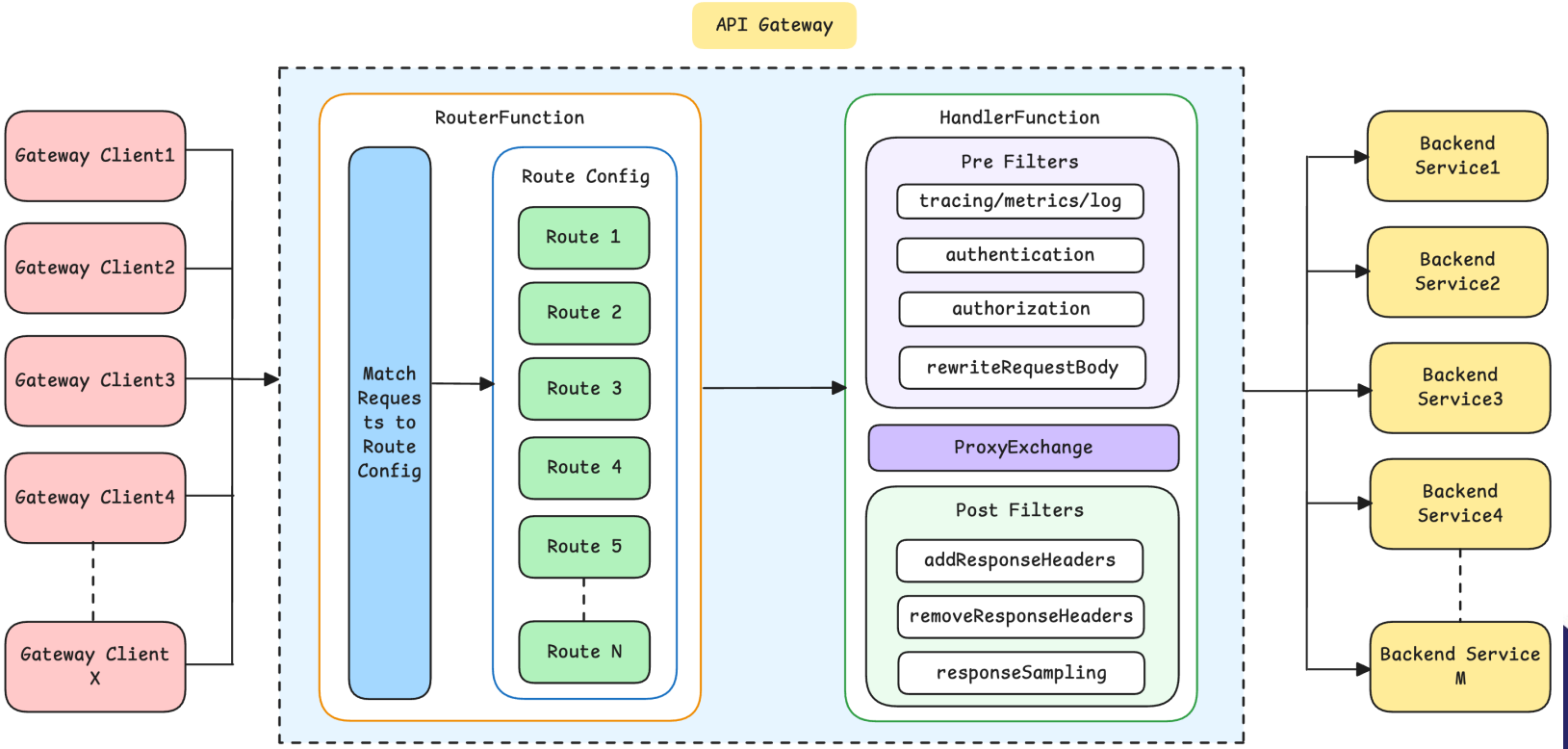
```
tracing()                                          // HandlerFilterFunction
  .andThen( metrics() )                            // HandlerFilterFunction
  .andThen( logging() )                            // HandlerFilterFunction
  .andThen( geetestVerify() )                      // HandlerFilterFunction
  .andThen( authentication() )                     // HandlerFilterFunction
  .andThen( authorization() )                      // HandlerFilterFunction
  .andThen( rewriteRequestBody() )                 // HandlerFilterFunction
  .andThen( retry(3) )                             // HandlerFilterFunction
  .andThen( lb( “jobservice” ) )                   // HandlerFilterFunction
  .apply( http() )                                 // HandlerFunction
```

Composing a Request Processing Pipeline

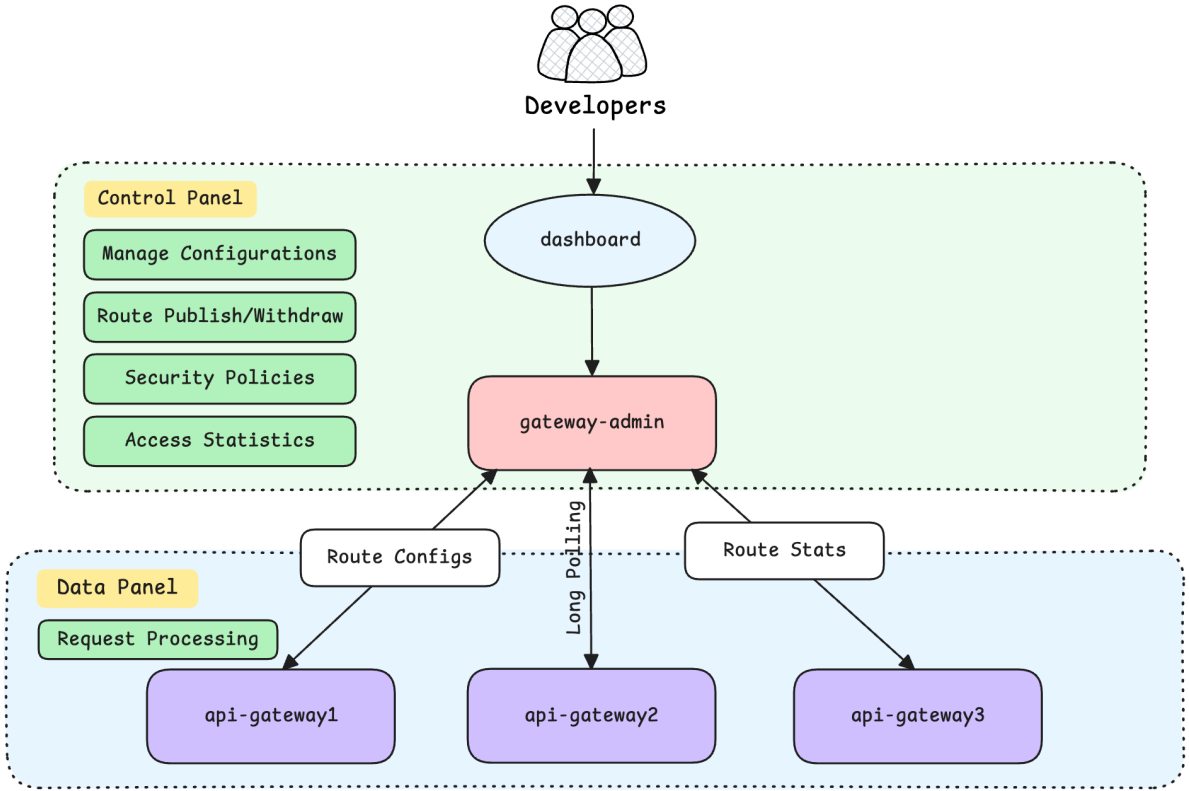
```
tracing()
  .andThen( metrics() )
  .andThen( logging() )
  .andThen( geetestVerify() )
  .andThen( bapiSession(opts) )
  .andThen( authorization(opts) )
  .andThen( rewriteRequestBody(opts) )
  .andThen( retry(3) )
  .andThen( lb( “jobservice” ) )
  .apply( http() )
```

```
host: api.zhaopin.com
path: /jobs
methods:
  - GET
bapiSession:
  enabled: true
  checkStaffReview: true
  checkStaffPenelty: true
rewriteRequestBody:
  contentType: application/json
  rules:
    - source: session.staffId
      target: requestBody.staffId
```

Data Plane Request Flow



Control Plane and Data Plane



Why Separate Control Plane and Data Plane?

Separation of Concerns

- Clear boundaries between config management and request execution
- Is it a config problem or a traffic problem?

Scalability & Stability

- Data plane stays lightweight, avoids business logic bloat
- Scale control and data planes independently

Deployment Flexibility

- Easier to isolate failure

COMMUNITY
THE ASF CONFERENCE
CODE

PART 03

Engineering Considerations

Fault Tolerance Design

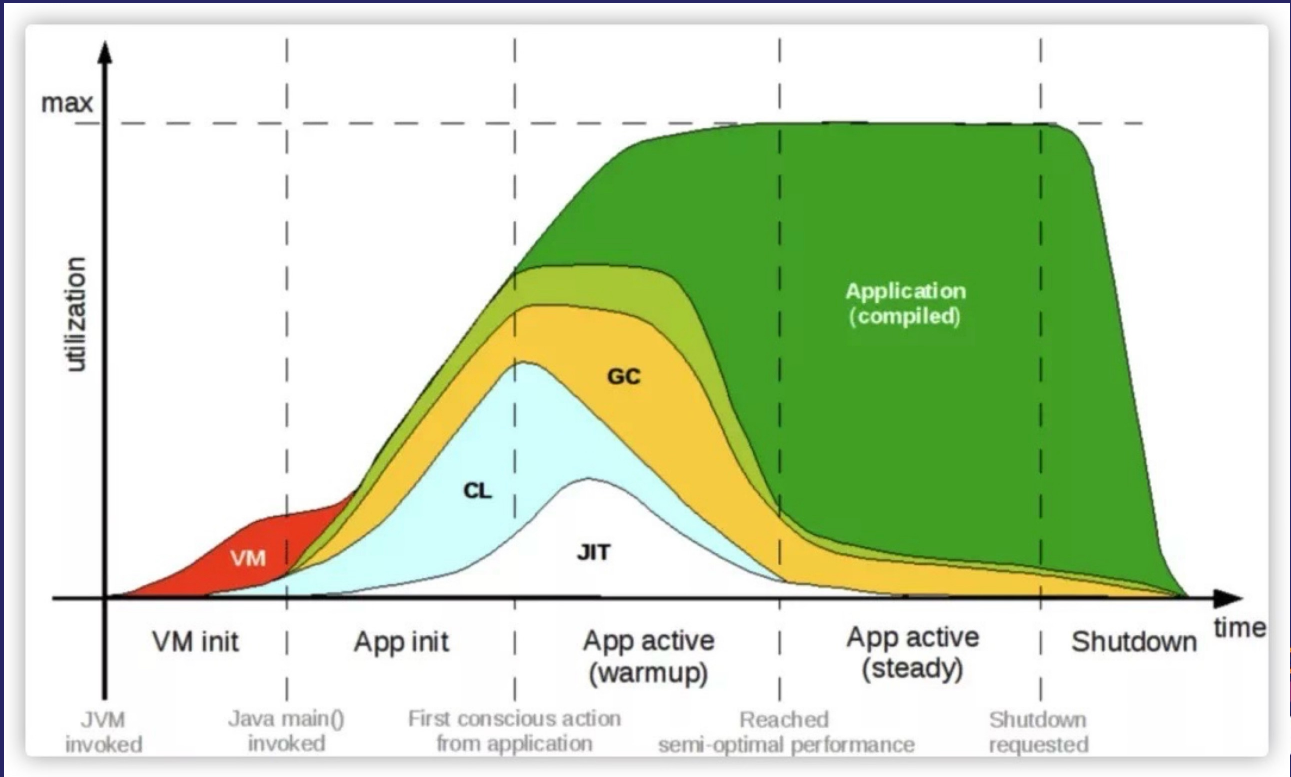
Retry Mechanism

- Safe retries on ConnectExceptoin, HttpConnectTimeoutException
- Do not blindly retry on SocketTimeoutException

Quarantized Zone for Target Isolation

- When ResourceAccessException occurs
- Temporarily isolate the failing target
- Prevent further impact on system health

Warmup-Aware Load Balancing



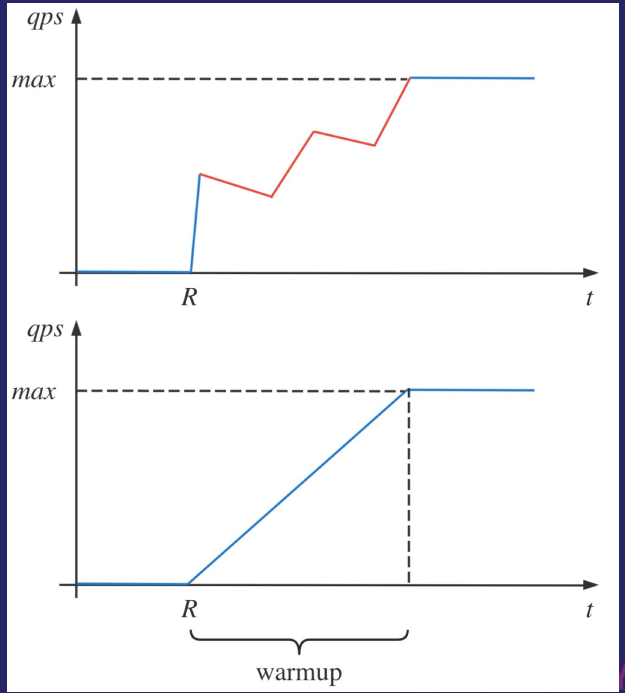
Warmup-Aware Load Balancing

Dynamic Weight Adjustment

- Use R-Point (Service Ready Timestamp)
- Calculate time offset since readiness
- Use it as a weight factor in load balancing

Smooth QPS Growth During Warmup

- Newly started instances receive gradually increasing traffic
- Prevents cold instances from being overloaded too soon
- Ensures stable and safe traffic ramp-up



Observability

Key Metrics

- Measures gateway’ s pre-processing overhead
- Tracks post-processing time
- Monitors number of active in-flight requests

Loging

- Mechanism to record request’ s end-to-end processing details

Trace

- OpenTelemetry Javaagent

COMMUNITY
THE ASF CONFERENCE
CODE

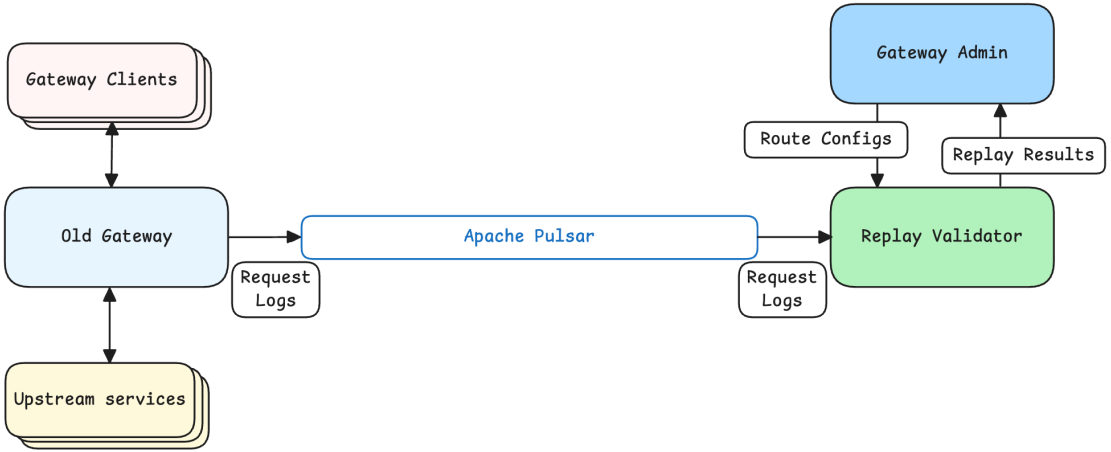
PART 04

Migration & Rollout

Migration Strategy

Step 1: Execution Record Collection

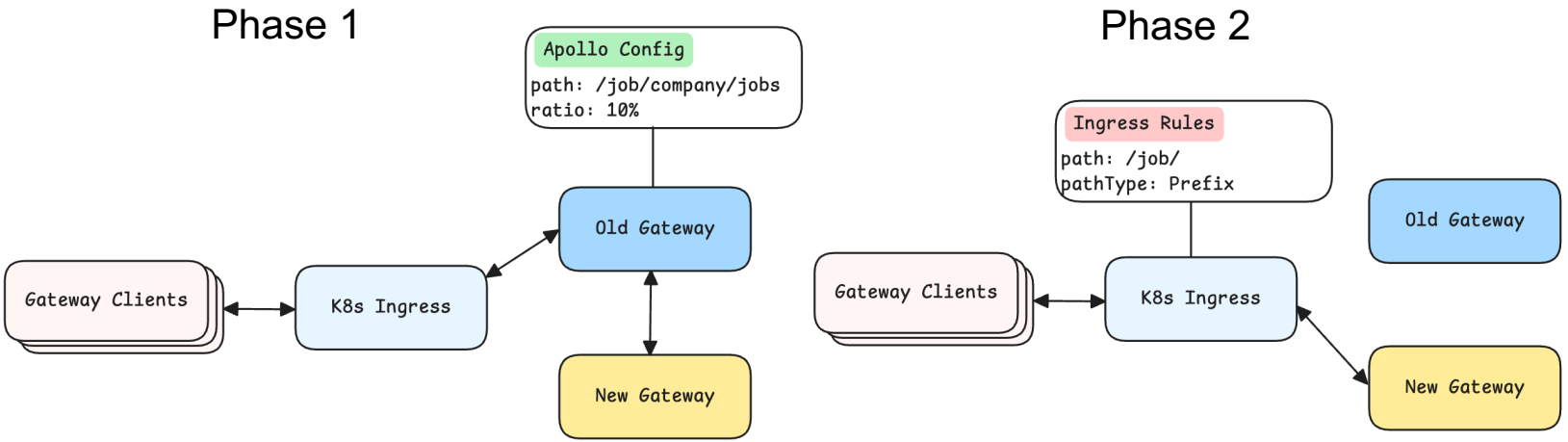
Step 2: Replay & Compare



Gradual Traffic Shift

Phase 1: Old gateway forwards some traffic to the new gateway

Phase 2: Ingress directly routes traffic to the new gateway

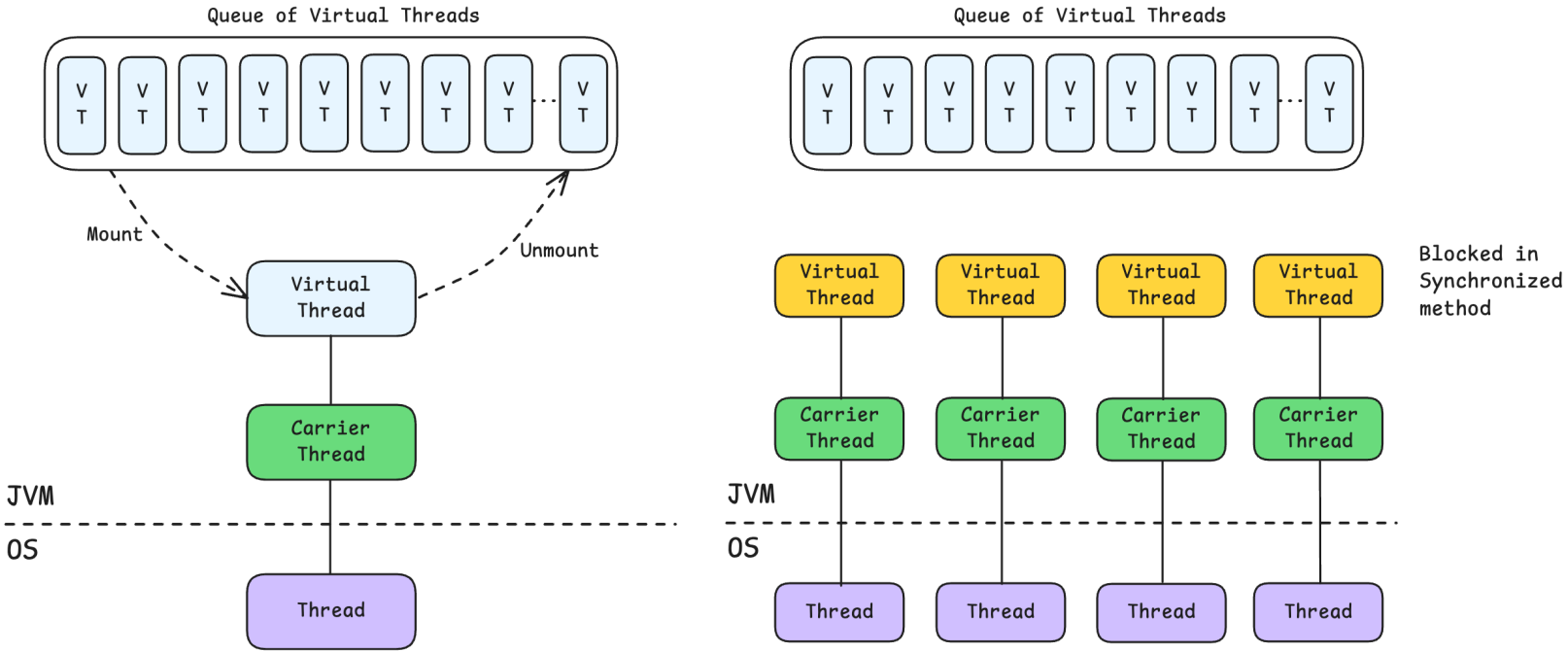


COMMUNITY
THE ASF CONFERENCE
CODE

PART 05

Challenges & Takeaways

Virtual Threads Pinning



Virtual Threads Pinning In Production (Java 21)

```
VirtualThread[#3080,tomcat-handler-1516]/runnable@ForkJoinPool-1-worker-4 reason:MONITOR
  java.base/java.lang.VirtualThread$VThreadContinuation.onPinned(VirtualThread.java:199)
  ...
  org.apache.http.pool.AbstractConnPool$2.get(AbstractConnPool.java:256) <== monitors:1
  org.apache.http.impl.client.InternalHttpClient.doExecute(InternalHttpClient.java:185)
  org.apache.http.impl.client.CloseableHttpClient.execute(CloseableHttpClient.java:83)
  org.apache.http.impl.client.CloseableHttpClient.execute(CloseableHttpClient.java:108)
  ...
```

Diagnosing Options

- -Djdk.tracePinnedThreads=full
- JFR Event: jdk.VirtualThreadPinned

JEP 491: Synchronize Virtual Threads without Pinning

Key Takeaways

- Build What Fits Your Context
- Functional Composition Simplifies the Pipeline
- Control Plane & Data Plane Separation Matters
- Handle Fault Tolerance Carefully
- Virtual Threads: Promising, but Use with Care
- Use Gradual Migration to Reduce Risk

COMMUNITY
THE ASF CONFERENCE
CODE

Thanks

Yike Xiao @shawyeok