APACHECON
ASIA 2022

Extending Valves in Tomcat

Dennis Jacob

2022. 07. 29

Biography

Dennis Jacob

- Senior Consultant @ Leading Payment Industry organization
- Part of Middleware Engineering Group
- Interests
  - Application Server Technologies
  - Cloud Native Technologies
  - Web application security

https://www.linkedin.com/in/dennis-jacob

https://github.com/dennisjacob

Agenda

- Request Processing Challenges
- Tomcat Valves
- Case Study – mTLS Valve
- Case Study – Tomcat Request Rate Limiter Valve
- Case Study – Debugging Request Processing Valve
- Caveats and Performance considerations
- Summary

Request Processing Challenges

Pre-processing of requests

Filtering the requests

Enforcing security

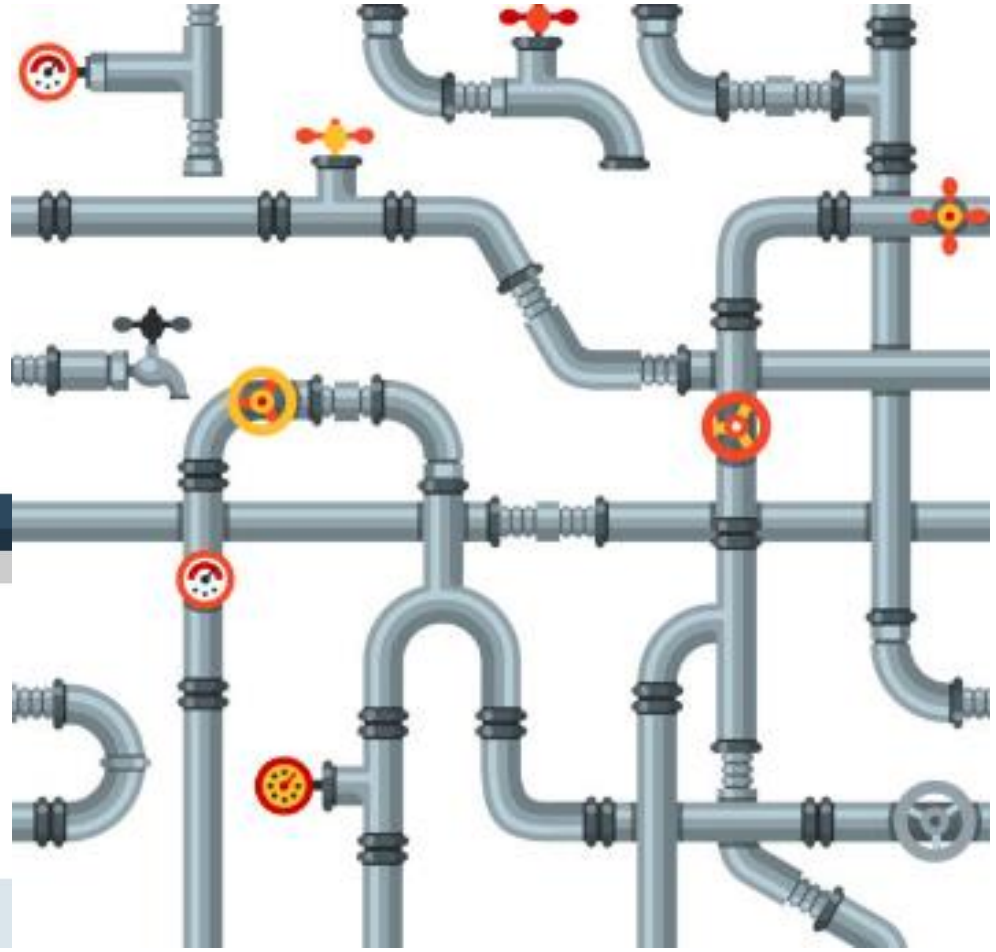
Traffic Rate Limiting

Troubleshooting

Monitoring and Logging

Solution??

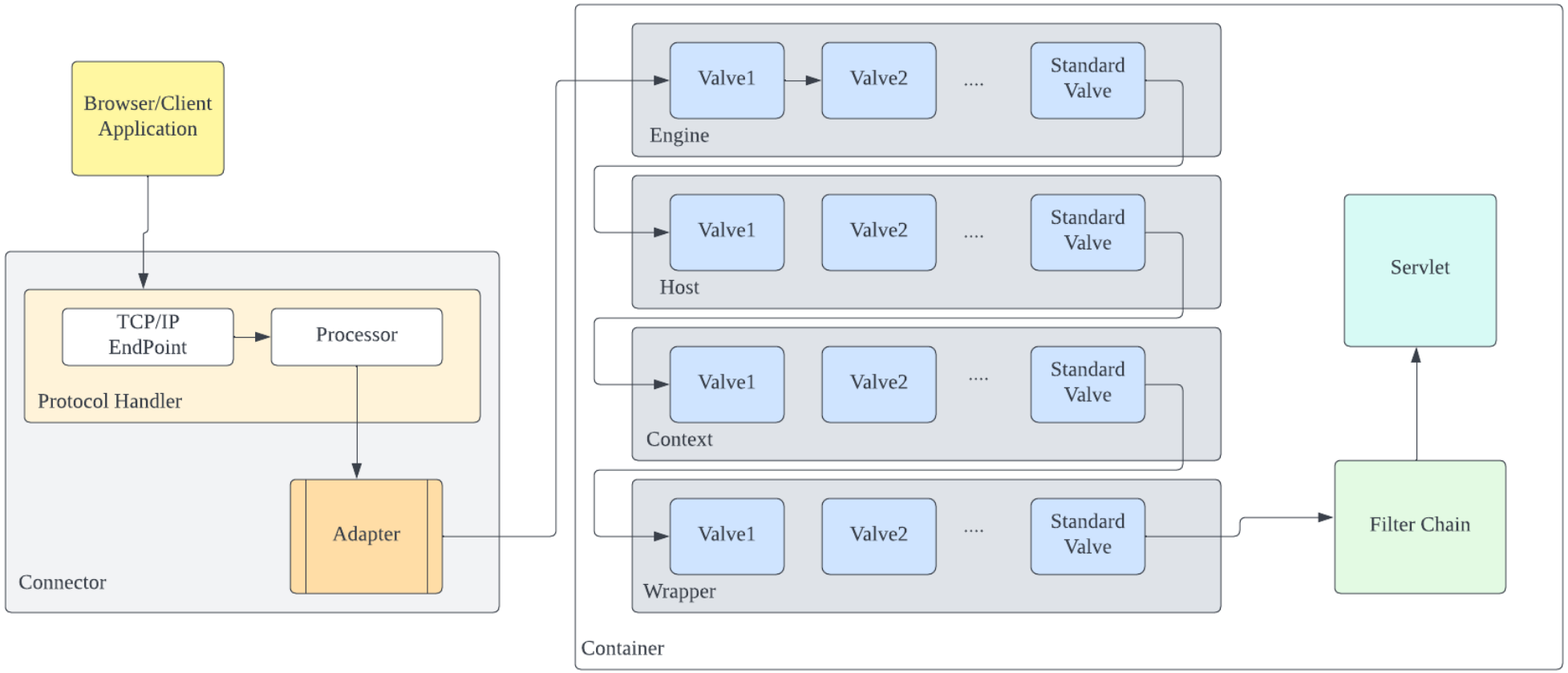
Tomcat Valve



Valves in Tomcat

- Request Processing pipeline in Tomcat
- What is Valve?
- How valves work in Tomcat?
- Application of Valves at Engine, Host or Context
- Valves and request pre-processing
- Valves vs Filters

Tomcat Request Processing Pipeline



Internal working of Valves

- ValveBase class
- Valves execution in pipeline
- How it helps in the request pre-processing?
- Customizing Valves

- java.lang.Object
  - org.apache.catalina.util.LifecycleBase
    - org.apache.catalina.util.LifecycleMBeanBase
      - org.apache.catalina.valves.ValveBase

Internal working of Valves (contd..)

Engine

```
final class StandardEngineValve extends ValveBase {
@Override
   public final void invoke(Request request, Response response) throws IOException, ServletException {
      Host host = request.getHost();
      host.getPipeline().getFirst().invoke(request, response);          } }
```

Host

```
final class StandardHostValve extends ValveBase {
@Override
   public final void invoke(Request request, Response response) throws IOException, ServletException {
      Context context = request.getContext();
       context.getPipeline().getFirst().invoke(request, response);          } }
```

Context

```
final class StandardContextValve extends ValveBase {
@Override
   public final void invoke(Request request, Response response) throws IOException, ServletException {
      Wrapper wrapper = request.getWrapper();
      wrapper.getPipeline().getFirst().invoke(request, response);        } }
```

Wrapper

```
final class StandardWrapperValve extends ValveBase {
@Override
   public final void invoke(Request request, Response response) throws IOException, ServletException {
      StandardWrapper wrapper =  (StandardWrapper) getContainer();
      Servlet servlet = null; Context context = (Context) wrapper.getParent();
     // Servlet Processing and calling filterChain
  } }
```

Commonly used Valves

- Access Logging Valves
- Access Control Valves
- Authentication Valves
- Error Report Valves
- Stuck Thread Detection Valve
- HealthCheck Valve
- Persistent Valve

Custom Valves

- How do we customize?
- Dependent libraries
- Creating Custom Valves
  - Define dependencies
  - Extend the ValveBase with custom requirement logic included
  - Building the artifact/jar
  - Using the jar in Tomcat classpath/lib
  - Configure Valve with attributes in Tomcat configuration

```
<dependencies>
    <dependency>
      <groupId>org.apache.tomcat.embed</groupId>
      <artifactId>tomcat-embed-core</artifactId>
      <version>9.0.59</version>
    </dependency>
    <dependency>
      <groupId>com.google.guava</groupId>
      <artifactId>guava</artifactId>
      <version>30.1.1-jre</version>
    </dependency>
```

```
public class CustomValve extends ValveBase {
    // define the getters and setters

@Override
public final void invoke(Request request, Response response) throws
ServletException, IOException {
                    // custom requirement code
                    getNext().invoke(request,response);  }}
```

```
<Valve className="com.org.dep.CustomValve" attribute="value" />
```

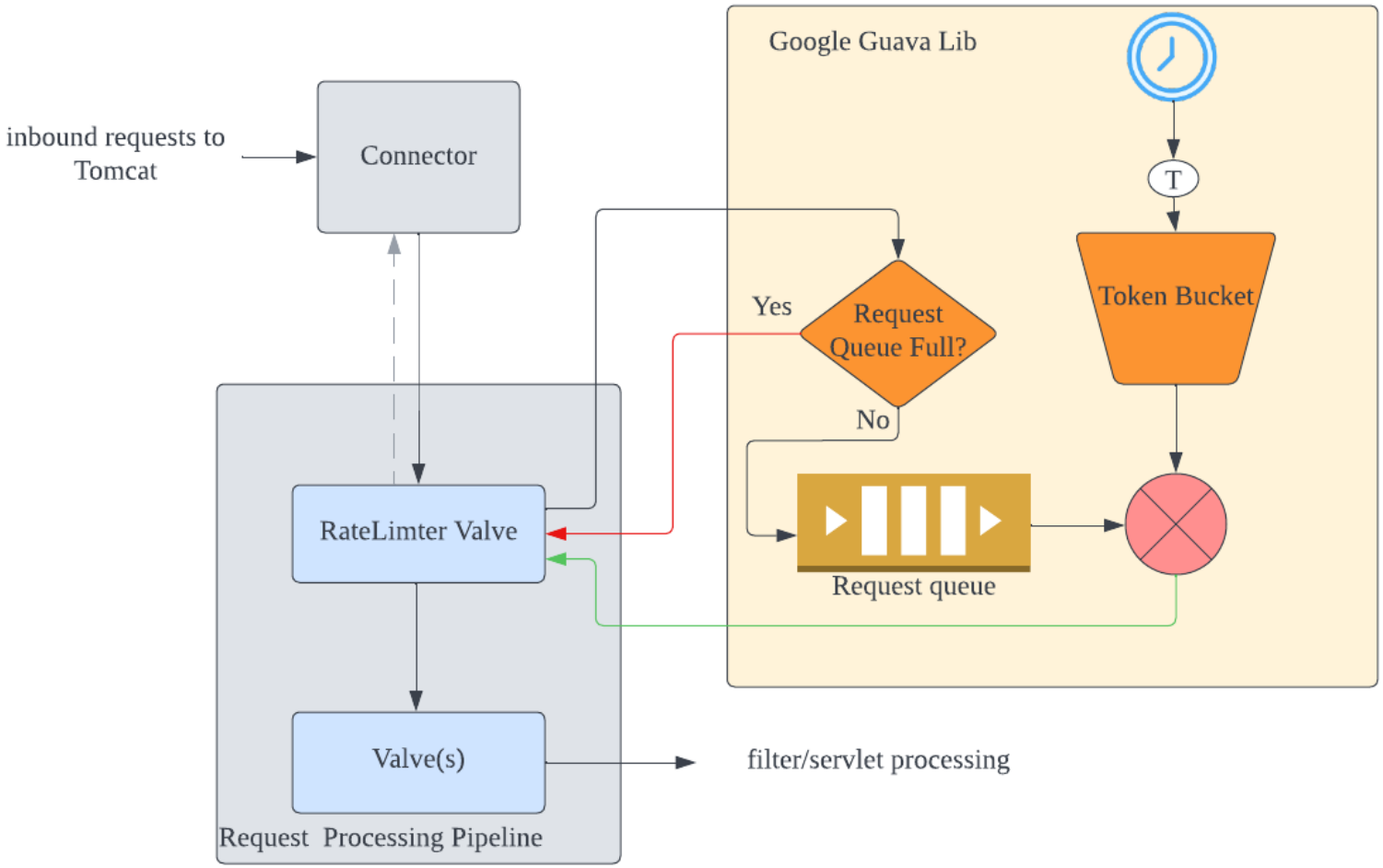
Use cases

Request Rate Limiter Valve

- Objective
  - Inbound Request Traffic throttling
  - Handling requirements for dynamic request rate throttling
  - Dependency on Load Balancer / Proxy Servers / Service Mesh

- Solutioning with Rate Limiter Valve
  - Using Google Guava API Library
  - Using Token Bucket Algorithm
  - Supports *Smooth Bursty* and *Smooth Warming Up* algorithms
  - Dynamic request rate throttling with a controller request
  - Further enhancement capabilities to build a controller plane

Request Rate Limiter Valve

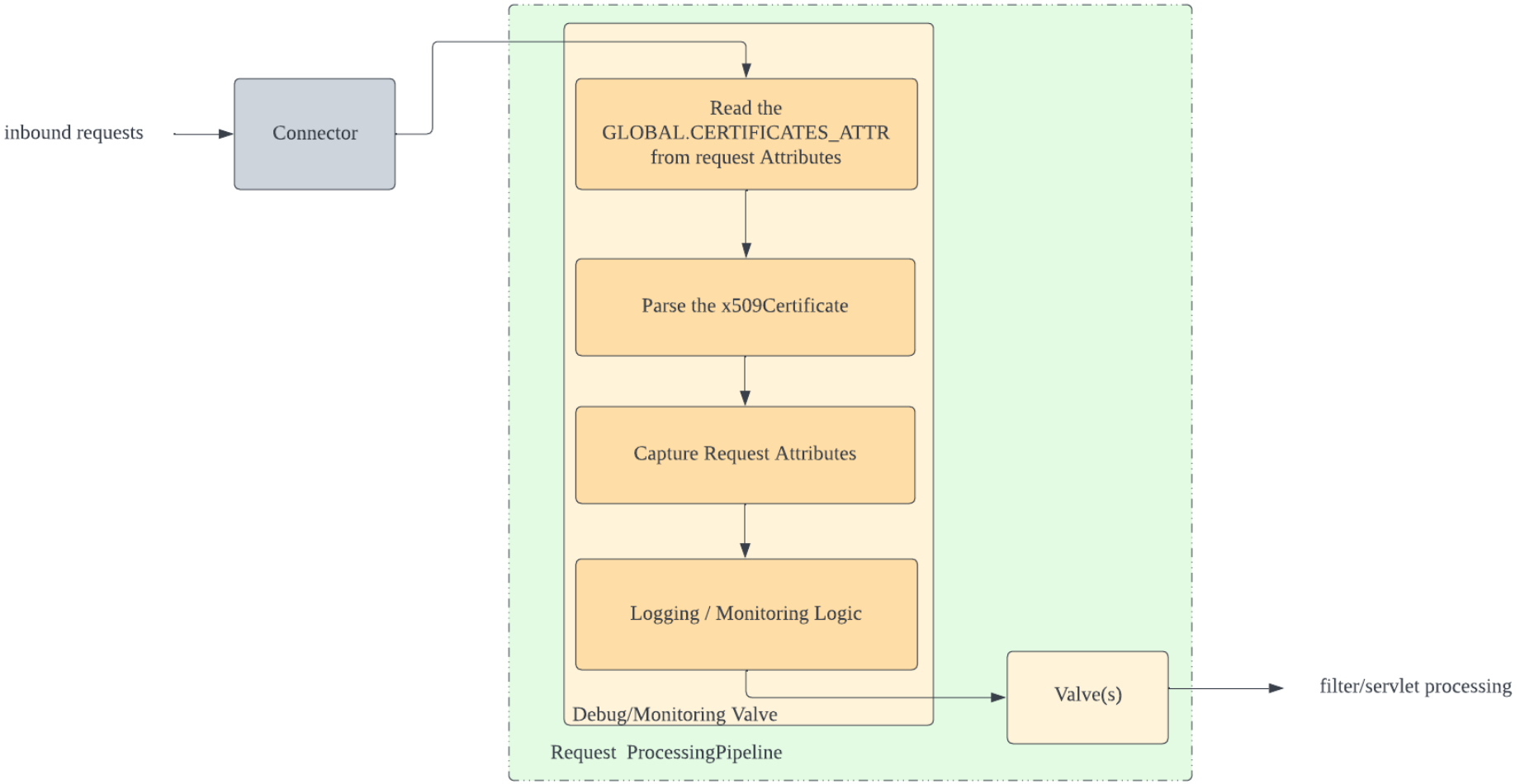


Request Debugger Valve

- Objective
  - Troubleshooting request/response attributes
  - Selective request debugging and logging
  - Checking Certificate attributes for 2-way TLS
  - Traffic replication for debugging
  - User defined header injection
  - Thread state and resource monitoring
  - Capturing JMX runtime bean attributes

- Solutioning with Debugger Valve
  - Read and parse the request/response attributes.
  - Get the certificate attributes for 2-way TLS
  - Capture JMX runtime bean attributes
  - Capture Thread states
  - Logging and routing the requests (selectively, if needed)
  - Sending the metrices to external monitoring systems

Request Debugger Valve

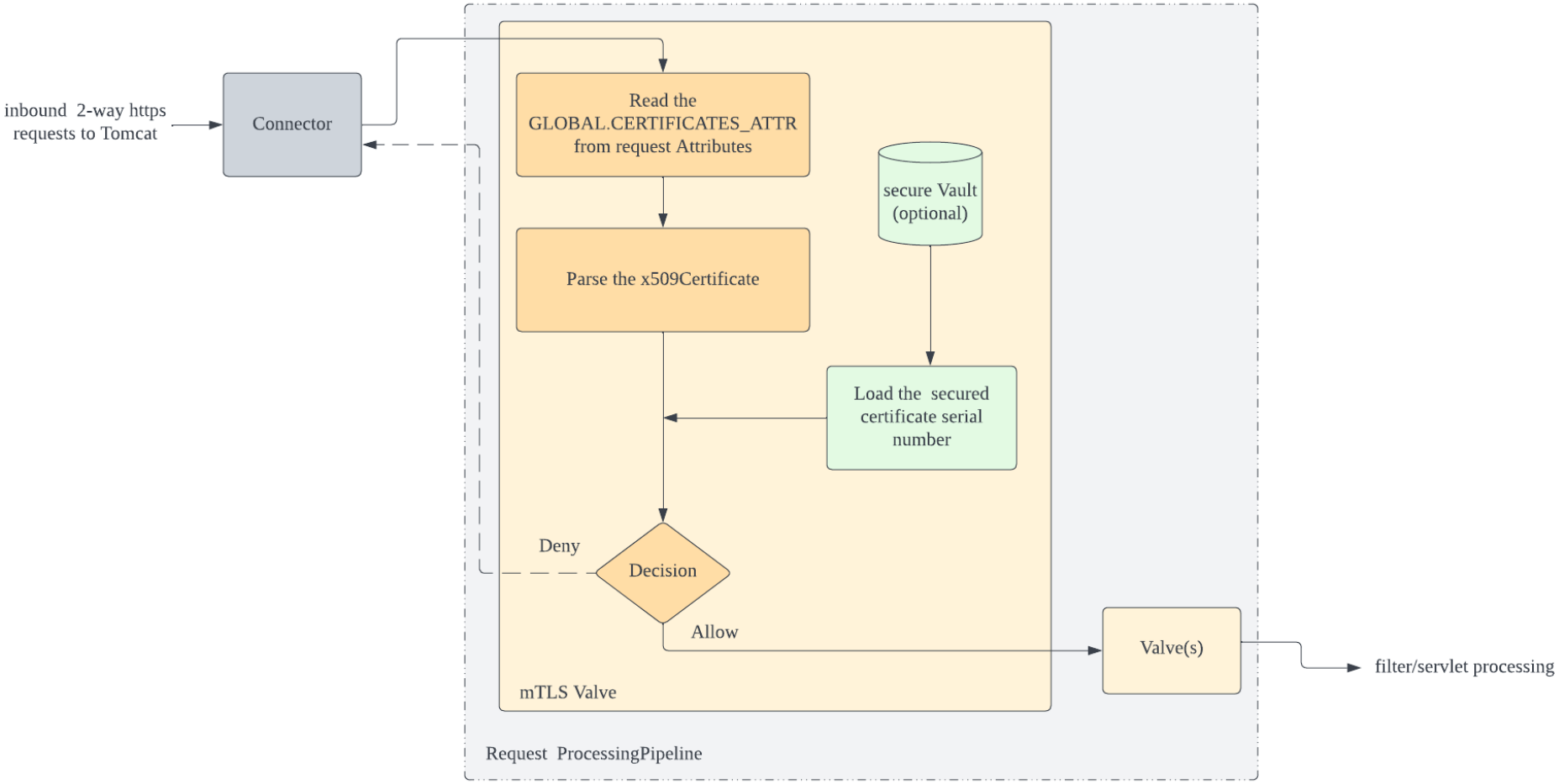


mTLS Valve

- Objective
  - Limitations of Client Certificate Verification with 2-way TLS
  - Complexities with Realms
  - Monitoring and troubleshooting
  - Performance challenges

- Solutioning with mTLS Valve
  - Pre-requisites
  - Whitelisting based on certificate serial number and DN
  - Parsing and logging the certificate attributes
  - Allow/Deny the request based on serial number and DN validation

mTLS Valve


inbound 2-way https requests to Tomcat
Connector
Read the GLOBAL.CERTIFICATES_ATTR from request Attributes
secure Vault (optional)
Parse the x509Certificate
Load the secured certificate serial number
Deny
Decision
Allow
Valve(s)
filter/servlet processing
mTLS Valve
Request ProcessingPipeline

Caveats and Performance considerations

- Performance considerations
  - Non-optimized usage of third-party libraries
  - Latency due to many Valves
  - Poor coding practices
  - Latency with feeds to external systems
- Caveats
  - Limitations on request processing outside the pipeline
  - Valves in Clustered Tomcat instances

Take aways

- Valves play a key role in Tomcat’s request processing pipeline.
- Valves can be created with custom logic, that perform extended logic implementation or checks at the request processing pipeline.
- Custom Valves can be developed and implemented, extending the ValveBase.
- Pay attention to performance optimization when developing custom valves.

Thank you

Questions?

APACHECON
ASIA 2022