Anypoint Platform Architecture: Application Networks

Module 7
Architecting and Deploying Effective API Implementations

Objectives

- Describe **auto-discovery** of API implementations implemented as Mule apps
- Appreciate how **Connectors** serve System APIs
- Describe **CloudHub**'s features and technology architecture
- Choose **Object Store** in a CloudHub setting
- Apply strategies that help API clients guard against **failures in API invocations**
- Describe the role of **CQRS** and the separation of commands and queries in API-led connectivity
- Explain the role of **Event Sourcing**

Developing API implementations for Mule runtimes and deploying them to CloudHub

Introducing API implementations management on Anypoint Platform

MuleSoft

API implementations may be

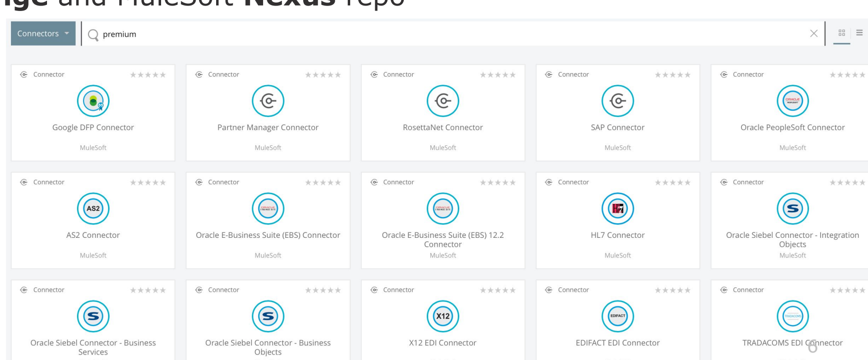
- Developed as **Mule apps** for the Mule runtime
    - **Studio** or **Flow designer**
    - Deployed to an Anypoint Platform **runtime plane** such as the **CloudHub**
    - API Management, Analytics, etc. provided by an Anypoint Platform control plane **with or without separate API proxies**
- Developed **for other runtimes** (Node.js, Spring Boot, etc.)
    - Deployed to matching runtimes outside Anypoint Platform
    - API Management, Analytics, etc. provided by an Anypoint Platform control plane via **API proxies executing in a runtime plane**

Introducing Auto-discovery of API implementations

- Auto-discovery is
  - **Auto-registration** of an API implementation
    - Developed as Mule app and deployed to Mule runtime (incl. API proxies)
  - With **API Manager**
- **On start-up** registers with API Manager as an implementation of a given API instance (implying API, version and environment)
- **Receives all API policies** for that API instance
  - By default refuses API invocations until all API policies have been applied
    - Gatekeeper feature

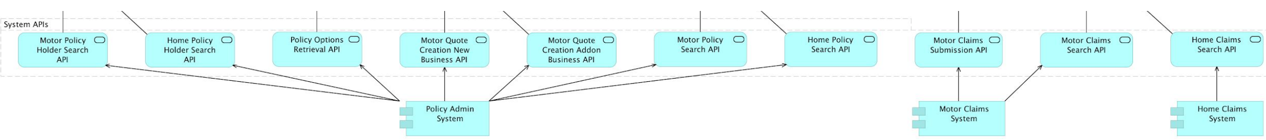
Anypoint Connectors to implement System APIs

- Connectors are **pre-built** (Anypoint Connectors) or **custom-developed** components that integrate Mule apps with external systems
  - More than **120** pre-built, many bundled with **Studio**
  - Others in **Exchange** and MuleSoft **Nexus** repo
  - **Support** categories:
    - Premium
    - Select
    - Certified
    - Community

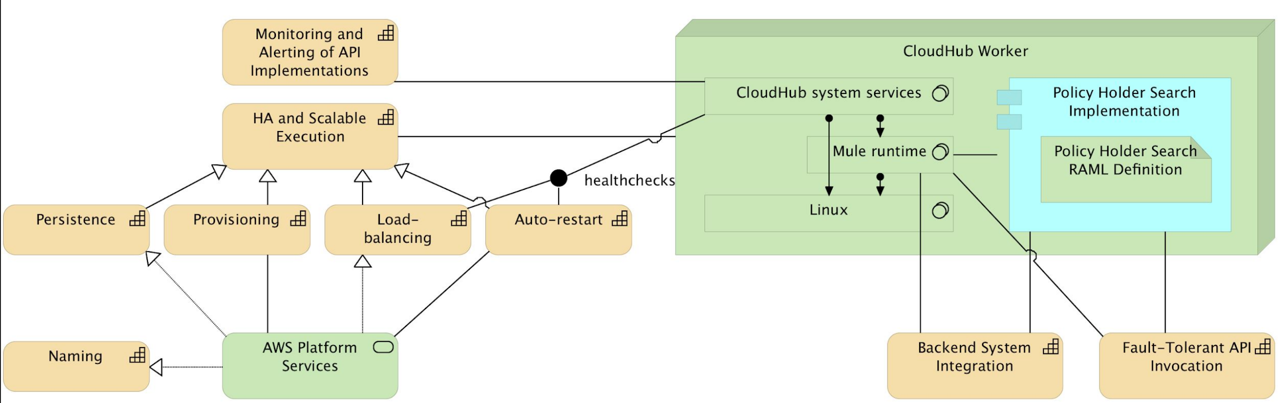


Connectors to implement System APIs

- Mainframe-based **Policy Admin System**
  - Accessed via **WebSphere MQ** using **JMS Connector**
- WebSphere-deployed **Motor Claims System**
  - Directly accessing its **DB/2** database using **Database Connector**
- Web-based **Home Claims System**
  - Exposes **SOAP APIs** accessed using **Web Service Consumer**



Fundamentals of CloudHub Technology Architecture



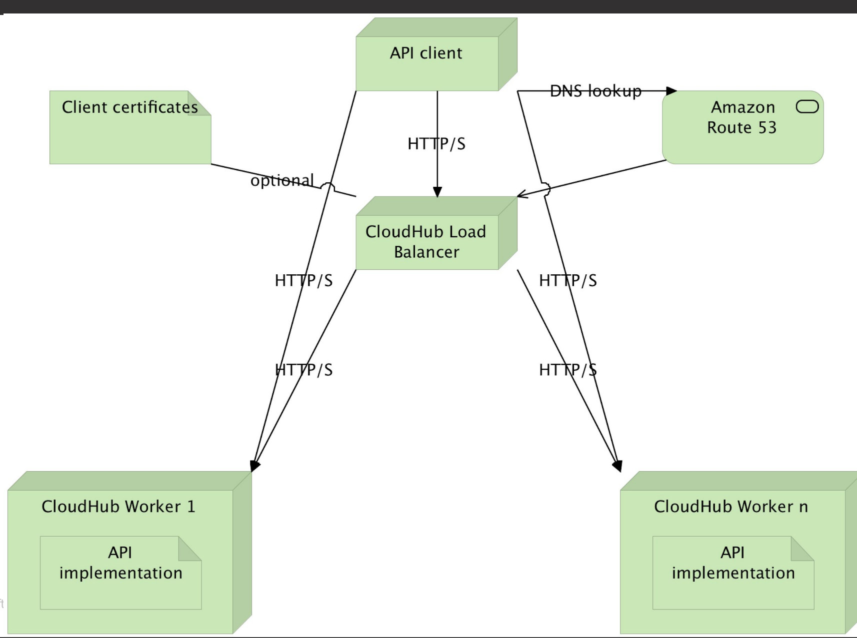
CloudHub worker sizing and illustrative mapping to EC2 instance types

MuleSoft

Worker Name	Worker Memory	Worker Storage	EC2 Instance Name	EC2 Instance Memory
0.1 vCores	500 MB	8 GB	t2.micro	1 GB
0.2 vCores	1 GB	8 GB	t2.small	2 GB
1 vCores	1.5 GB	8 + 4 GB	m3.medium	3.75 GB
2 vCores	3.5 GB	8 + 32 GB	m3.large	7.5 GB
4 vCores	7.5 GB	8 + 40 + 40 GB	m3.xlarge	15 GB
8 vCores	15 GB	8 + 80 + 80 GB	m3.2xlarge	30 GB
16 vCores	32 GB	8 + 160 + 160 GB	m4.4xlarge	64 GB

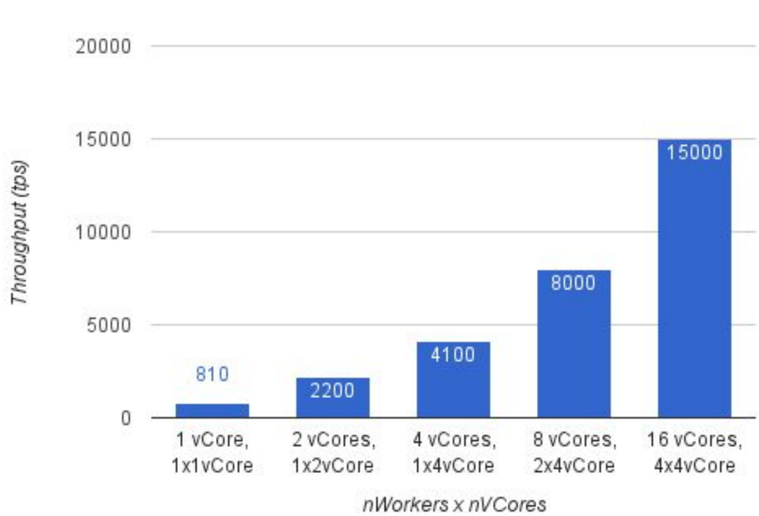
Fundamentals of CloudHub Technology Architecture

MuleSoft



Performance of CloudHub workers of various sizes

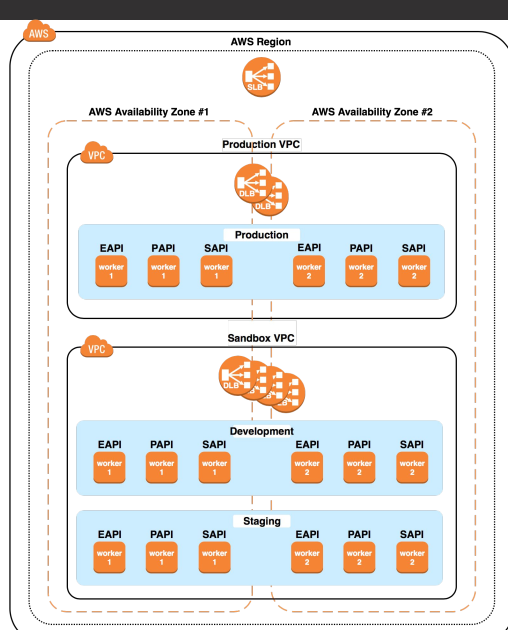
MuleSoft



- Throughput as HTTP requs/s
- of a simple API implementation

CloudHub Shared Worker Cloud and Anypoint VPC

MuleSoft



CloudHub Shared Worker Cloud and Anypoint VPC

MuleSoft

Environments | Business Groups | **Firewall Rules** | Internal DNS

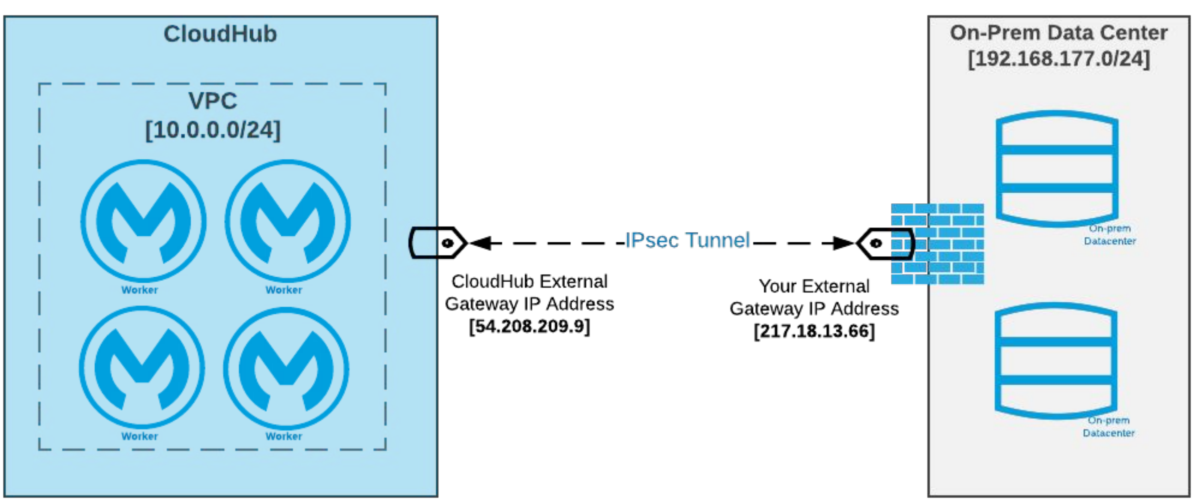
Learn more about Firewall Rules

⊕ Add New Rule

Type	Source	Port Range	
https.private.port	Local VPC (10.2.0.0/16)	8092	...
https.port	Anywhere (0.0.0.0/0)	8082	...
http.private.port	Local VPC (10.2.0.0/16)	8091	...
http.port	Anywhere (0.0.0.0/0)	8081	...

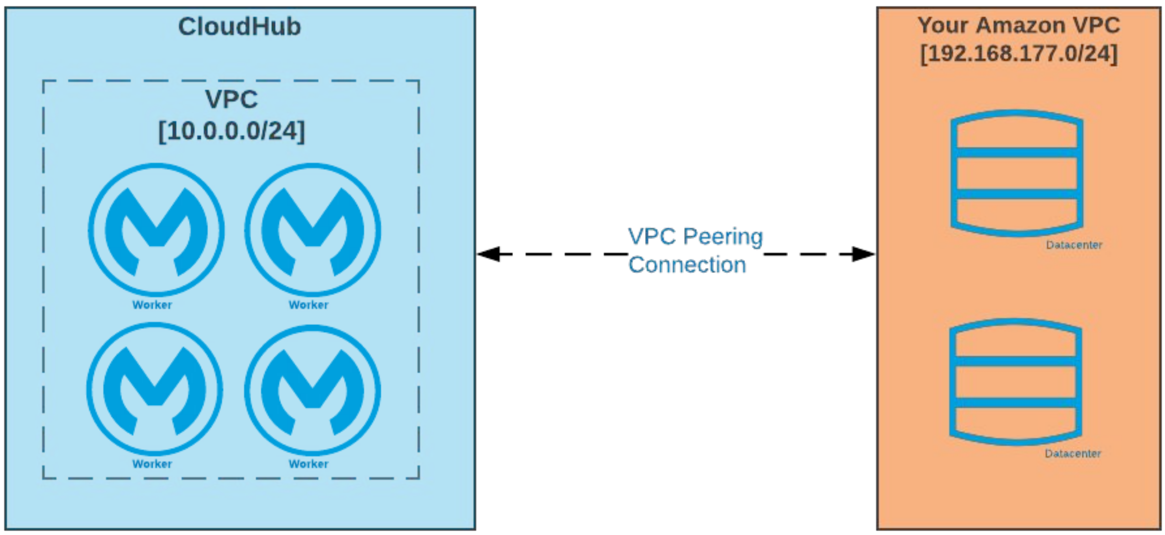
Anypoint VPC connectivity: on-premises networks

MuleSoft



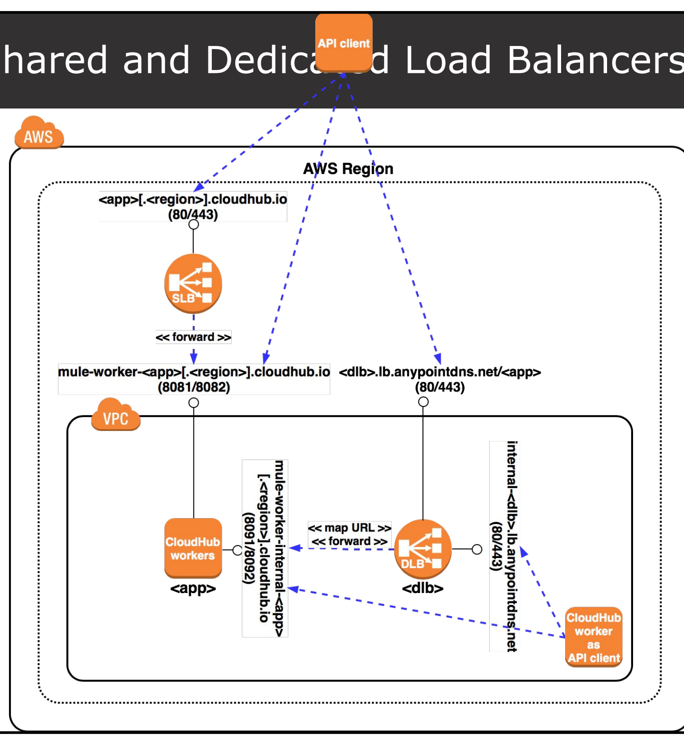
Anypoint VPC connectivity: AWS VPC peering

MuleSoft



CloudHub Shared and Dedicated Load Balancers

MuleSoft



CloudHub DNS names

	Control Plane Region	
	US East	**EU (Frankfurt)**
Web UI **Platform APIs**	anypoint.mulesoft.com	**eu1.**anypoint.mulesoft.com
DLB Public IPs	<dlb>**.lb.**anypointdns.net	<dlb>**.lb-prod-eu-rt.**anypointdns.net
DLB Private IPs	**internal-**<dlb>**.lb.**anypointdns.net	**internal-**<dlb>**.lb-prod-eu-rt.**anypointdns.net

CloudHub DNS names for some runtime plane regions

mule-worker- mule-worker-internal-	Control Plane Region	
	US East	**EU (Frankfurt)**
US East	<app>.cloudhub.io	N/A
EU (Frankfurt)	<app>**.eu.**cloudhub.io	<app>**.de-c1.eu1.**cloudhub.io
EU (Ireland)	<app>**.eu.**cloudhub.io	<app>**.ir-e1.eu1.**cloudhub.io
US West (Oregon)	<app>**.us-w2.**cloudhub.io	N/A
EU (London)	<app>**.uk-e1.**cloudhub.io	N/A
Canada	<app>**.ca-c1.**cloudhub.io	N/A
Singapore	<app>**.sg-s1.**cloudhub.io	N/A
Sydney	<app>**.au-s1.**cloudhub.io	N/A

Exercise: CloudHub for Acme Insurance

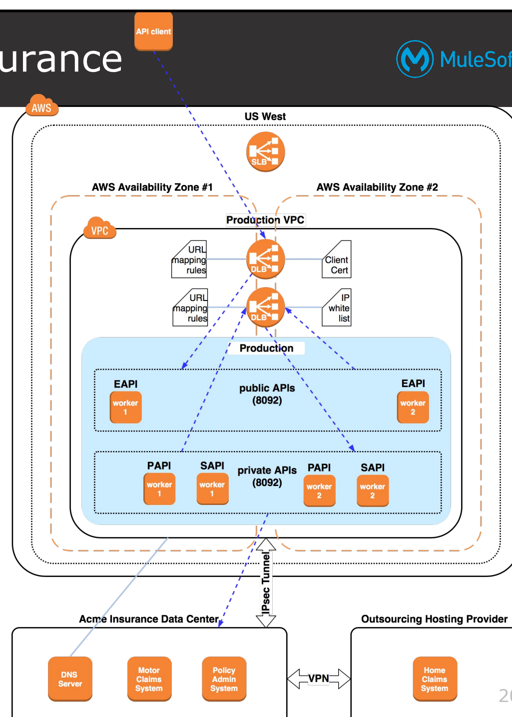
MuleSoft

Acme Insurance will deploy all their API implementations to **CloudHub**, under the **US control plane**. Based on your knowledge of their requirements (making assumptions if necessary) sketch their **Production environment**:

1. Region(s) of their Anypoint Platform runtime plane(s)
2. Number of workers per API implementation and worker size
3. Need for and use of Shared Worker Cloud, Anypoint VPCs, Shared/Dedicated Load Balancers, IPsec tunnels, VPC peering
4. API URL patterns, in particular hostnames, as seen by API clients
5. Ports on which API implementations listen

Solution: CloudHub for Acme Insurance

MuleSoft



External API clients:
https://api.acmeinsurance.com/aggregatorquotecreation/ -> ans-aggregatorquotecreation-eapi

VPC-internal API clients:
https://private-api.acmeinsurance.com/policyoptionsranking-papi/ -> ans-policyoptionsranking-papi

Understanding Object Store

- OS provides **key-value persistence** to all Mule apps
- Use for:
  - **correlation information** for async processing, circuit breakers, **caches**, …
  - **idempotent** filters/validators, **watermarks**, **token stores**, stateful API policies such as **Rate Limiting**, …
- Object Store **Connector**
- Mule app can use **several OS instances**
  - **No inter-Mule app data exchange**
- Values can also be **JSON** documents
- **No transactions or queries**
- **Persistent** or **transient**, **TTL**, max capacity
- **Default persistent OS** always available

Understanding Object Store in CloudHub

- Keeps data in **same AWS region** as the owning Mule app
  - Available (only) to **all workers** of the owning Mule app
- OS content exposed in **Runtime Manager**
- Max **TTL of 30 days**
- **Unlimited** number of **keys**; each **value** limited to **10 MB**
- **Cross-AZ HA** within an Amazon Web Services region
- **Encryption** in transit and at rest

Designing API clients to use fault-tolerant API invocations

Exercise: Failures in API invocations

MuleSoft

1. List every kind of failure that can occur when an API client invokes an API implementation
2. Think of ways of guarding against each of the failures you have identified

Solution: Failures in API invocations

The invocation of an API implementation by an API client fails if any of the **intervening components** fails at the moment of the API invocation:

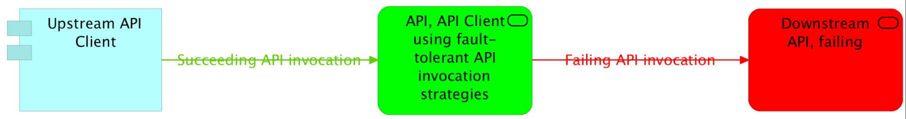
- **Hardware**
- **Virtualization** or **container** stacks
- **Operating system**, **JVM** or other runtime and **libraries**
- Any **network component** such as Ethernet cards, switches, routers, cables, WIFI transmitters, …
- Includes all **Anypoint Platform components** and underlying **AWS** services: API policies, API proxies, load balancers, CloudHub workers, etc.
- The **API implementation** itself, because of a bug in the app code, a failed deployment, ...

Understanding the significance of failure in application networks

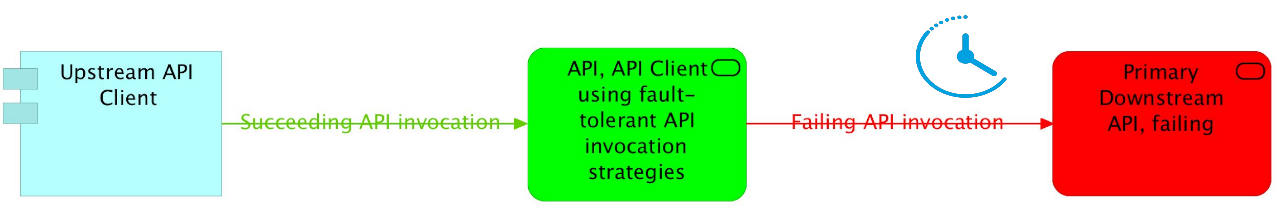
- High degree of **dependency** of any API implementation on other APIs
  - Inherent and **desired** feature of application networks
  - Shows high degree of **reuse** of APIs
- **Failures** in invocation of an API **affect many other APIs**
  - If unchecked, failures in API invocations propagate **transitively** through an application network
- Therefore: Make **API invocations fault-tolerant**

Designing API clients for fault-tolerant API invocations

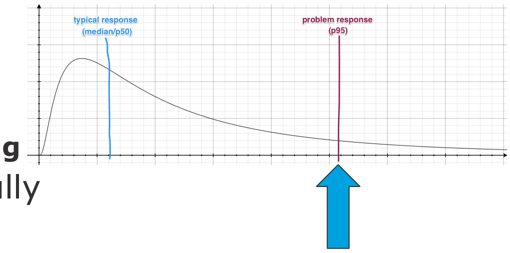
- **Fault-tolerant API invocation**:
  - **Invocation** from an API client to an API implementation via its API
  - Where **failure** of the API invocation
  - Does not necessarily lead to **failure** of the API client
  - API client is typically itself an API implementation
    - **Breaks transitive failure propagation**
- **Strategies**:
  - Timeout
  - Retry
  - Circuit Breaker
  - Fallback Invocation
  - Opportunistic Parallel Invocations
  - Fallback Result
  - Client-Side Cache



Using timeouts to guard against slow APIs



Using timeouts to guard against slow APIs

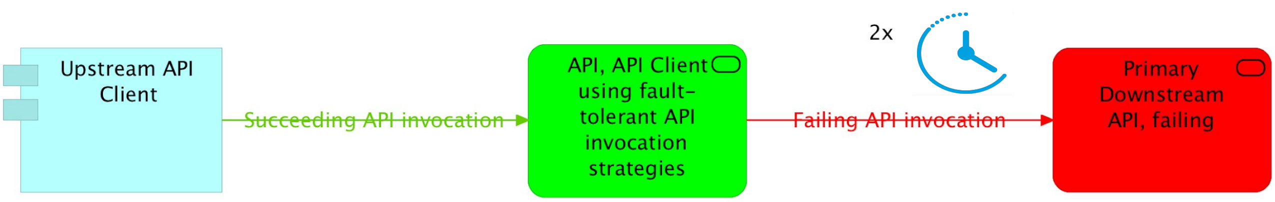


- **Slow** API invocations
  - Jeopardize **SLA** of API client at risk
  - Have higher probability of **eventually failing**
- Set **timeouts** for API invocations carefully
  - In accordance with the SLA of the API client
  - As **low** as possible, given SLA of invoked API
- Timed-out API invocation is indistinguishable from a **failed** API invocation
  - Requires other **additional** fault-tolerance strategies
- For **Mule apps**:
  - At the level of the **HTTP request**
  - For an entire **transaction**
  - As part of higher-level control constructs like **Scatter-Gather**

Using timeouts to guard against slow APIs

- "Aggregator Quote Creation EAPI" has SLA of median/max response time of **200 ms/500 ms**
- Synchronously invokes **3 Process APIs**, in turn, starting with "Policy Holder Search PAPI"
  - Task performed by "Policy Holder Search PAPI" is the simplest and **fastest** of all 3 Process APIs
  - "Aggregator Quote Creation EAPI" should define timeout of no more than approx. **100 ms** for the invocation of the "Policy Holder Search PAPI"
    - Most invocations (98%) must be guaranteed to complete within 100 ms

Retrying failed API invocations



Retrying failed API invocations

- Failed API invocation may **succeed if retried**
  - Only if failure was **transient**
- Difficult for API client to **identify** transient failures
- For **REST** APIs:
  - **4xx** response codes signify permanent failures, except
    - HTTP 408 Request Timeout
    - HTTP 423 Locked
    - HTTP 429 Too Many Requests
  - **5xx** response codes may signify transient failures
  - Only **idempotent** HTTP methods may be retried:
    - GET, HEAD, OPTIONS, PUT, DELETE

Retrying failed API invocations

- To **limit** processing time:
  - Only **few** retries
  - **Short** wait times between retries
  - Short **timeouts** for each API invocation
- For **Mule apps**:
  - **HTTP Request Connector**
    - Has configurable support for interpreting HTTP response codes as failures
  - **Until Successful Scope**

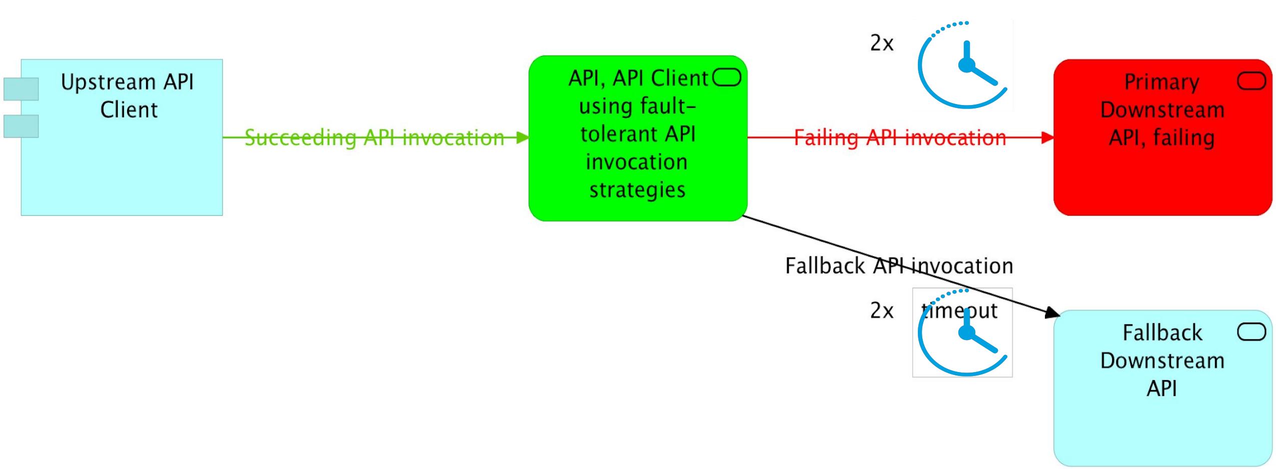
Invoking failing APIs less often with Circuit Breakers

- Failing API implementation may need time to **recover**
  - Continued invocations may **hinder** recovery
  - API client is **saved wasted time** and effort of invocating a failing API
- **Circuit Breaker**
  - **Monitors** API invocations
  - After "many" failed API invocations "**opens**"
    - Immediately fails invocations **without invoking** API
  - After "recovery period" **resumes** API invocations
    - Immediately "**opening**" upon failure
  - After "many" successful API invocations "**closes**"

Invoking failing APIs less often with Circuit Breakers

- Circuit Breaker is a **stateful** component
  - Keeps track of health state of API **for API clients**
- Varying **scope**/reach of API clients that experience same state:
  - One API client in a **single CloudHub worker**
  - All instances of an API client app in **all CloudHub workers**
    - Requires remote communication between instances of the API client
  - All instances of **any API client** app in any CloudHub worker
    - Circuit Breaker as an application network-wide shared resource
- For **Mule apps**:
  - **Open-source** implementations of the Circuit Breaker pattern

Invoking a fallback API



Invoking a fallback API

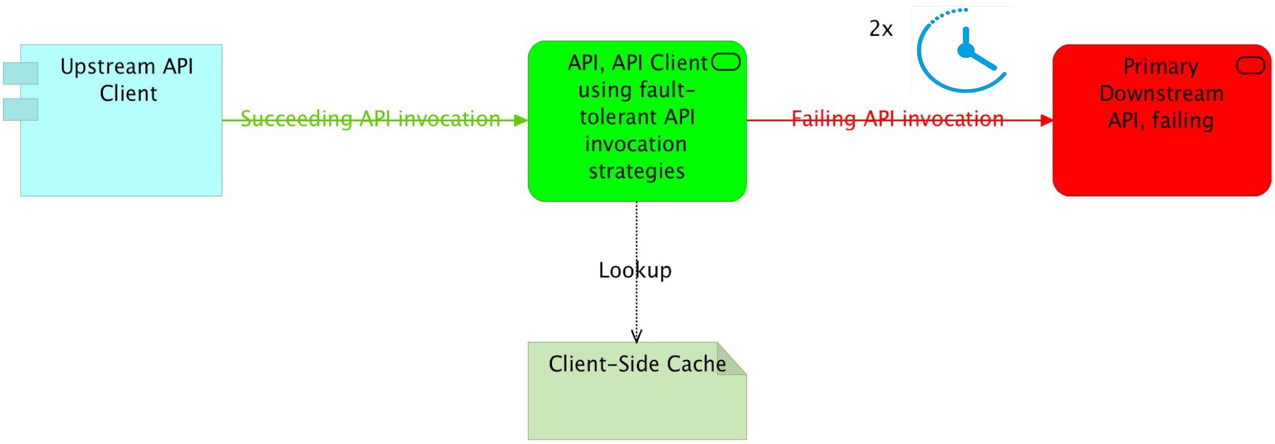
- After repeated API invocation failures it may be possible to invoke a **different API** as a fallback
  - May not be ideal but **sufficient**
  - Prefer **service degradation** over failure
- E.g., after repeated failures invoking "Motor Policy Holder Search SAPI" try:
  - Old but sufficiently compatible **version** of the same API
  - **Endpoint** of the same API from the DR site or other ClougHub region
  - "Motor Policy Search SAPI" which provides **functional super-set**
- For **Mule apps**:
  - **Until Successful Scope** plus exception strategies allow configuring fallback actions such as fallback API invocations

Opportunistically invoking APIs in parallel

- Invocation of primary API and fallback API may be performed in **parallel**
  - **First response** to arrive is used, the other discarded
- Reduces overall **execution time** compared to serial fallback invocation
- Opportunistic, **egotistical** strategy
  - Puts increased **load** on the application network
  - Only in **exceptional** cases

Using a previously cached result as a fallback

MuleSoft

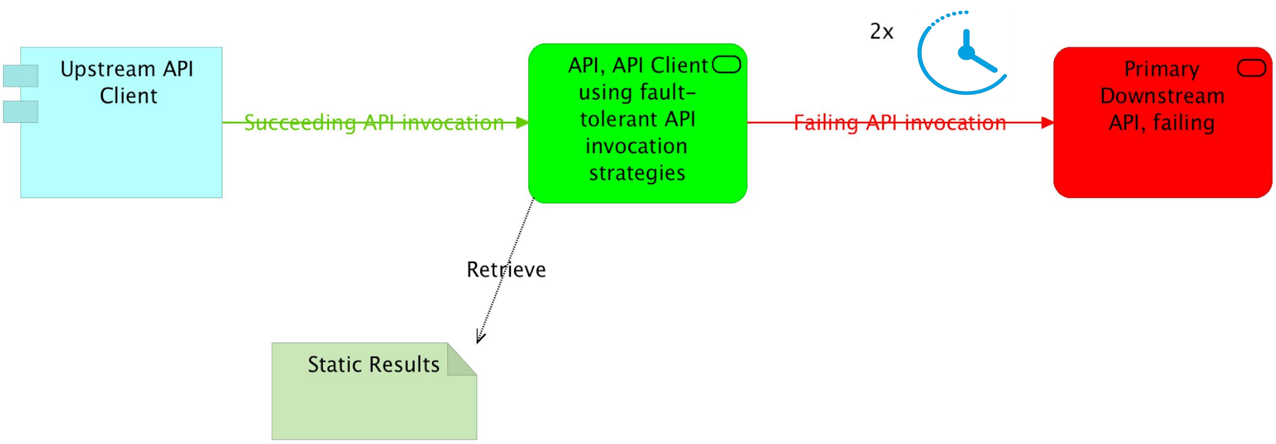


Using a previously cached result as a fallback

MuleSoft

- **Client-side caching** by the API client is a great source of fallback results
- Governed by the usual **HTTP caching** rules:
  - Only responses from **safe** HTTP methods should be cached:
    - GET, HEAD, OPTIONS
  - HTTP **caching headers** should be honored (but may not)
    - Cache-Control, Last-Modified, Age, …
- Increases **memory** footprint and processing of API client
- E.g., cached response from previous "Policy Options Retrieval SAPI" invocations may be used in case of failure
- For **Mule apps**:
  - **Cache Scope** and **Object Store Connector** support client-side caching

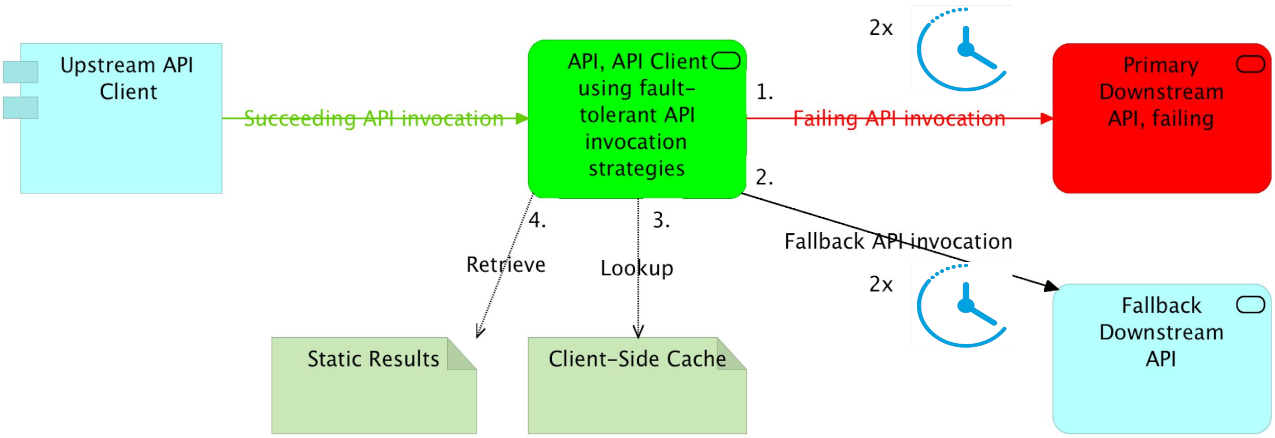
Using a static fallback result



Using a static fallback result

- **Prepared result** may be used instead of the result expected from an API invocation
- Best for APIs that return **reference data**-like results:
  - countries
  - states
  - currencies
  - products
- E.g., instead of result from "Policy Options Retrieval SAPI" a list of **common policy options** loaded from a configuration file:
  - **Not ideal** but better than not creating a quote at all
- For **Mule apps**:
  - Many options for results storage/retrieval, incl. **properties**

Designing API clients for fault-tolerant API invocations



Understanding architecturally significant persistence choices for API implementations

Introducing CQRS as an API implementation strategy

MuleSoft

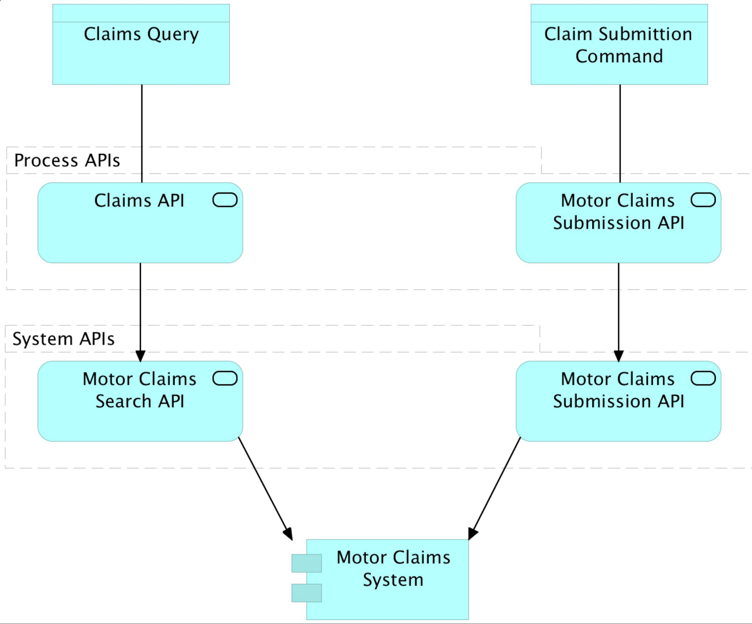
- **CQRS**
  - Command Query Responsibility Segregation
  - Usage of **different models** for
    - **reading** from data (queries) and
    - **writing** to data (commands)
- Reads and writes can be optimized and scaled **independently**
- Independent **persistence** mechanisms for reads and writes
- **Commands**
  - Formulated in the domain language of the **Bounded Context**
  - Trigger **writes**
  - Typically queued and executed **asynchronously**

Introducing CQRS as an API implementation strategy

MuleSoft

- **Queries**
  - Optimized for the **API clients**' needs (joins, aggregates)
  - Execute **synchronously**
  - May return slightly **out-of-date data**
- **Complicates** architecture and implementation of an API
- **Visible in API** itself
- Independent design choice **for each API implementation**
  - Architecturally **significant** because
    - Causes **eventual consistency** between read-sides and write-sides
    - Apparent in the **API specification**
    - May cause **API** to be **split** into one API for queries and one for commands

Viewing Acme Insurance's application network from a CQRS perspective



Introducing persistence with Event Sourcing

- **Event Sourcing**
  - Approach to data **persistence** that keeps persistent state as a series of **events** rather than as a snapshot of the current state
  - Often combined with CQRS
- Similar to **database transaction logs**, but in **application layer**
- **Events**
  - Expressed in the domain language of the **Bounded Context**
  - Typically arise directly from executing CQRS-style **commands**
    - **Propoagate state changes** from CQRS write-side to CQRS read-side
    - Relies on **asynchronous messaging** systems like Anypoint MQ
- **Invisible in API** specification: **implementation detail**

Summary

Summary

- Anypoint Platform-managed API implementations can target **Mule runtime or other runtimes**
- API implementations implemented as Mule apps can be **automatically discovered** by Anypoint Platform
- Anypoint Platform has over 120 **Connectors**
  - Indispensable for implementing **System APIs**
- **CloudHub** is an AWS-based iPaaS for the scalable, performant and highly-available deployment of Mule apps
- **Object Store** is a key-value persistence service available in all Mule runtime deployment scenarios

Summary

- **In CloudHub**, Object Store allows time-limited persistence local to the AWS region of the Mule runtime
- API clients must guard against **API invocation failures**
    - In particular those which are API implementations
    - **Retry, Circuit Breaker, Fallbacks**, ...
- Some API implementations may benefit from **CQRS**
- **Separation of commands and queries** often arises naturally in API design, even in absence of true CQRS
- **Event Sourcing** is an implementation-level design decision of each API implementation