

Where is my cache?

Architectural patterns for caching microservices by example

• • •

Rafał Leszko
 @RafalLeszko
Hazelcast

About me

- Cloud Software Engineer at Hazelcast
- Worked at Google and CERN
- Author of the book "Continuous Delivery with Docker and Jenkins"
- Trainer and conference speaker
- Live in Kraków, Poland



About Hazelcast

- Distributed Company
- Open Source Software
- 140+ Employees
- Hiring (Remote)!
- Recently Raised \$21M
- Products:
 - Hazelcast IMDG
 - Hazelcast Jet
 - Hazelcast Cloud



Agenda

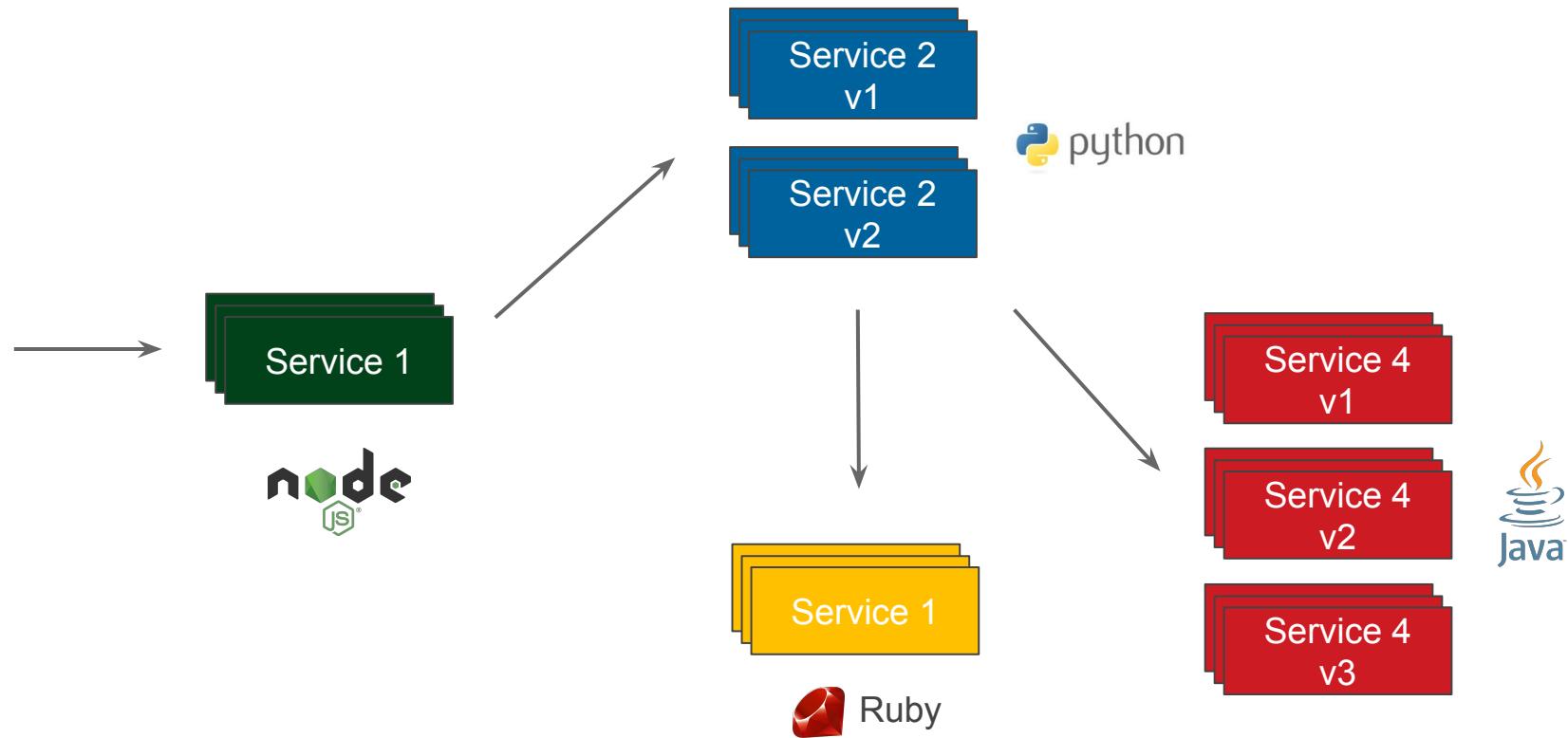
- Introduction
- Caching Architectural Patterns
 - Embedded
 - Embedded Distributed
 - Client-Server
 - Cloud
 - Sidecar
 - Reverse Proxy
 - Reverse Proxy Sidecar
- Summary

Why Caching?

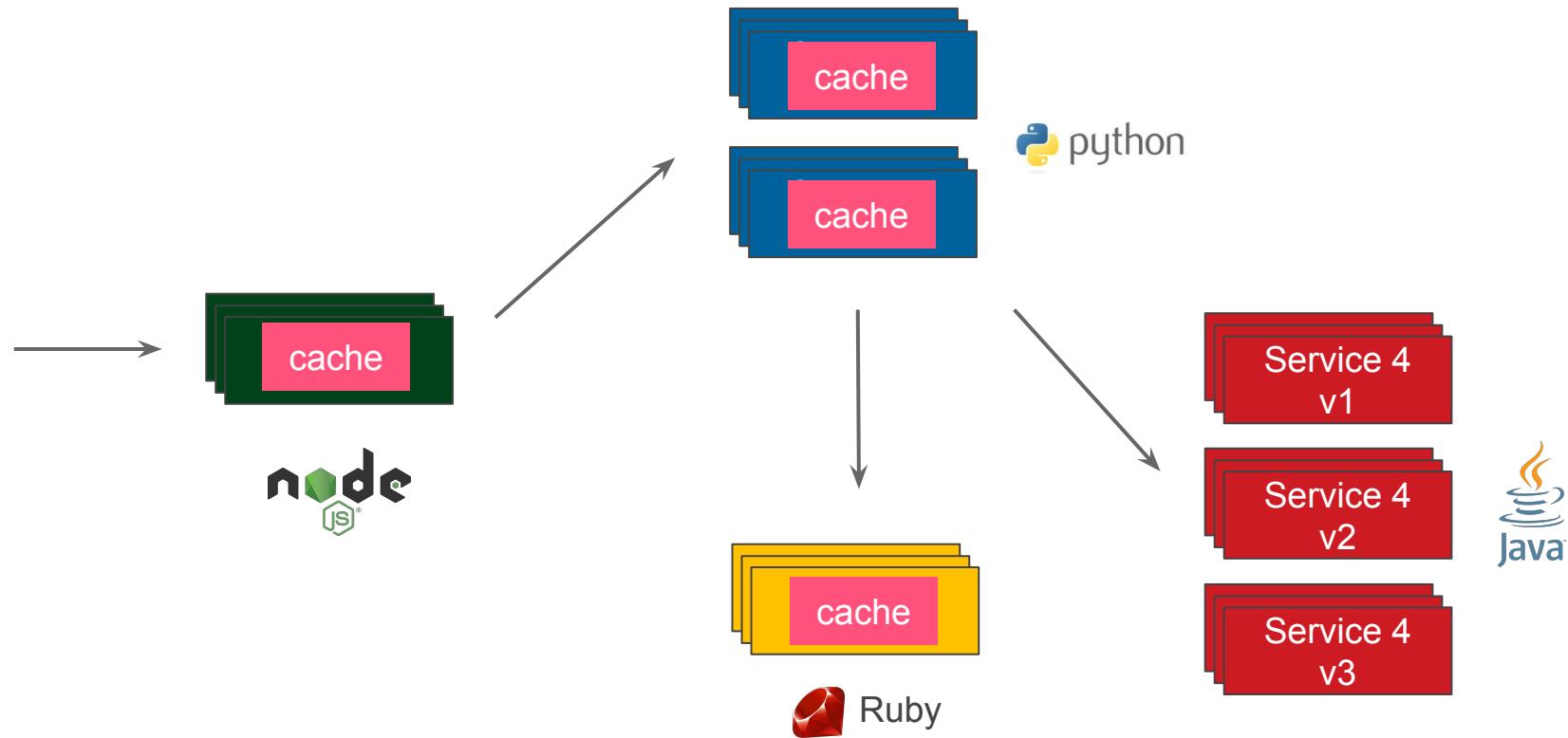
- Performance
 - Decrease latency
 - Reduce load
- Resilience
 - High availability
 - Lower downtime



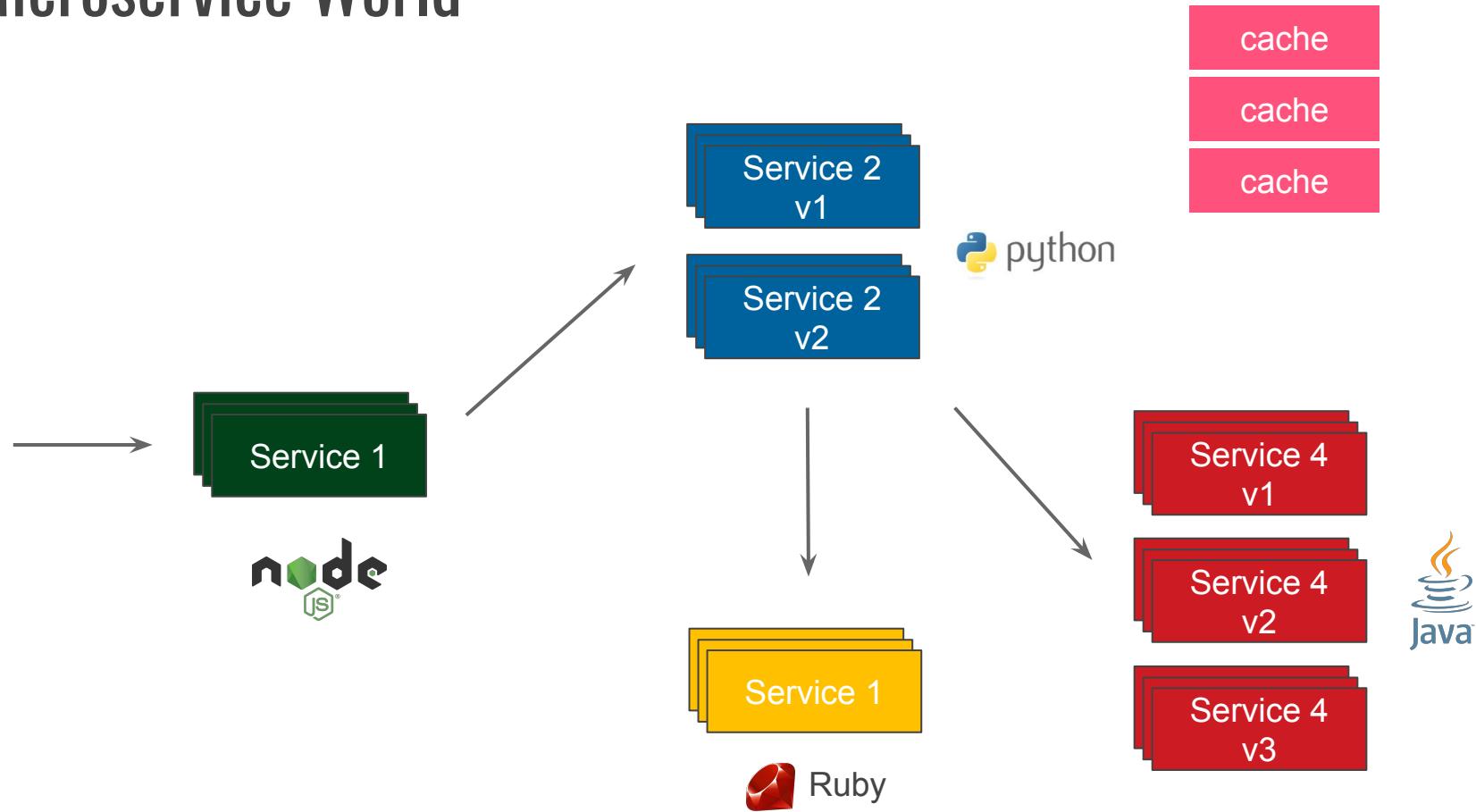
Microservice World



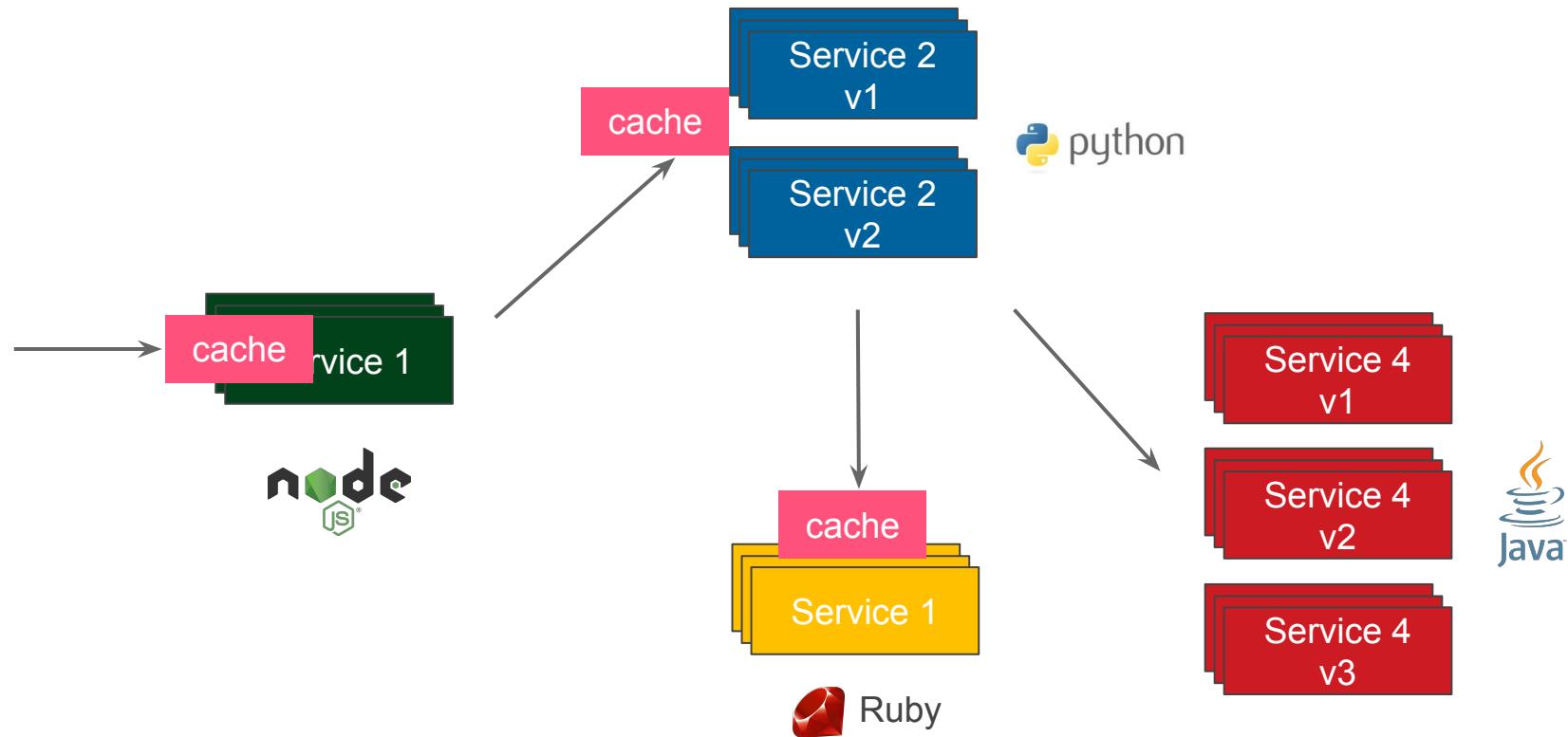
Microservice World



Microservice World

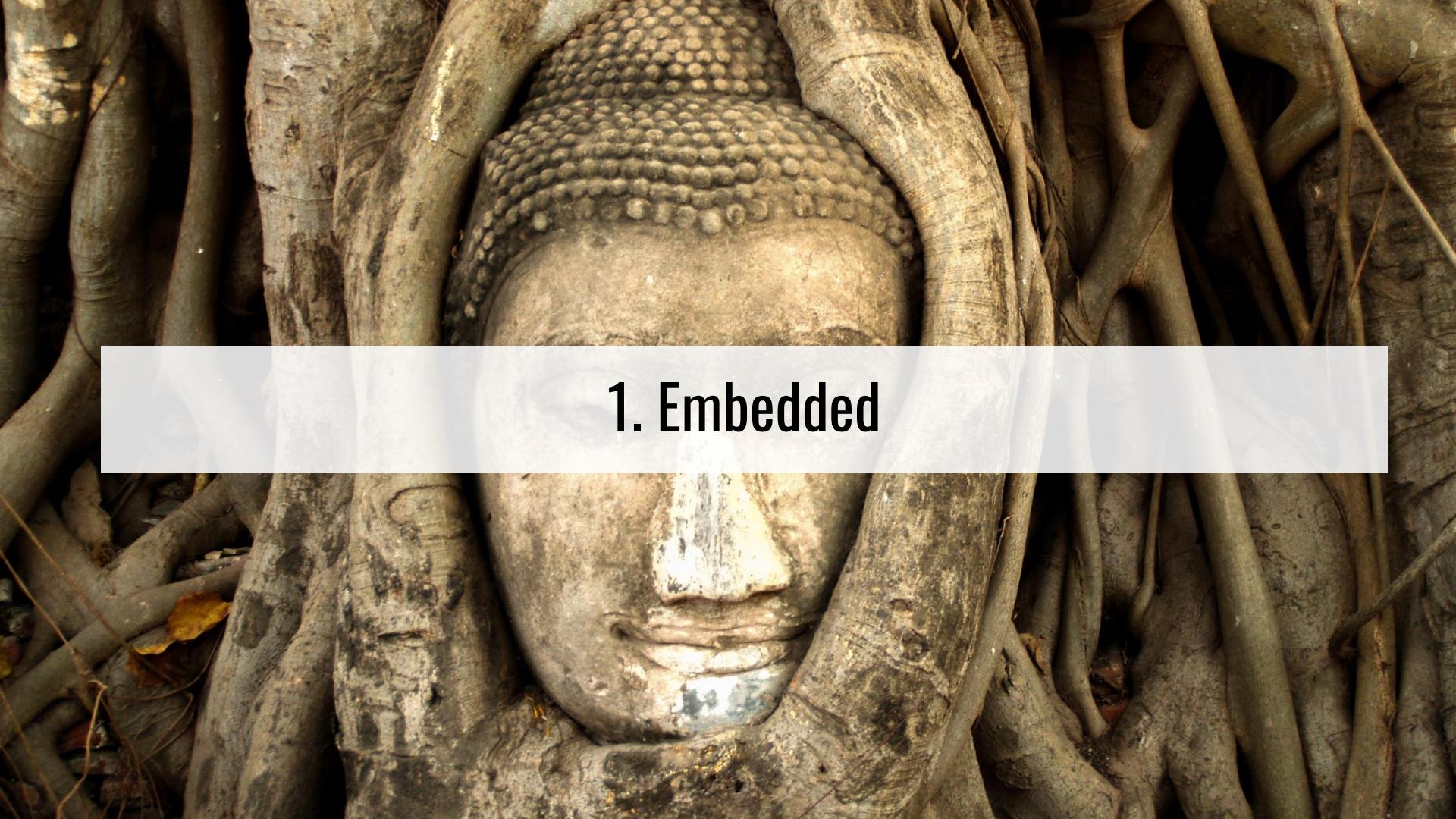


Microservice World



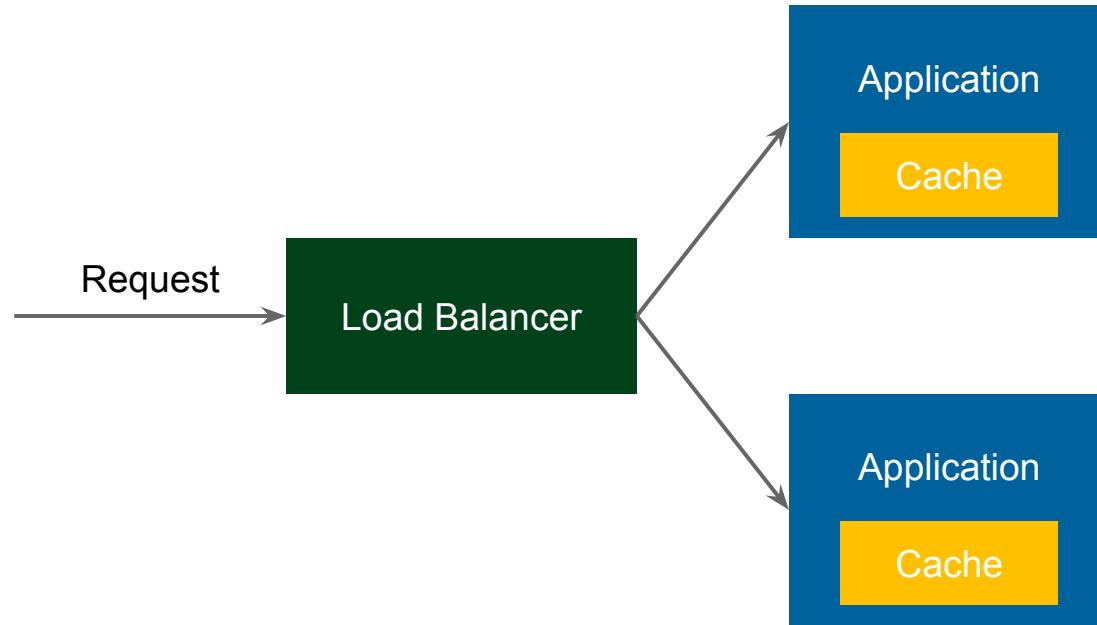
Agenda

- Introduction ✓
- Caching Architectural Patterns
 - Embedded
 - Embedded Distributed
 - Client-Server
 - Cloud
 - Sidecar
 - Reverse Proxy
 - Reverse Proxy Sidecar
- Summary

A close-up photograph of a large, weathered stone head of a Buddha, possibly made of sandstone or similar material. The head is heavily textured with numerous small, raised circular marks, likely representing a mottled skin pattern. It is partially embedded in the thick, gnarled roots of a large tree, specifically a Banyan tree, which are wrapped tightly around the stone. The tree's bark is dark brown and deeply fissured. The background is a dense thicket of similar trees. A white rectangular overlay covers the upper portion of the image, containing the text.

1. Embedded

Embedded Cache



Embedded Cache (Pure Java implementation)

```
private ConcurrentHashMap<String, String> cache =  
    new ConcurrentHashMap<>();  
  
private String processRequest(String request) {  
    if (cache.contains(request)) {  
        return cache.get(request);  
    }  
    String response = process(request);  
    cache.put(request, response);  
    return response;  
}
```

Embedded Cache (Pure Java implementation)

```
private ConcurrentHashMap<String, String> cache = new ConcurrentHashMap<>();  
  
private String processRequest(String request) {  
    if (cache.contains(request)) {  
        return cache.get(request);  
    }  
    String response = process(request);  
    cache.put(request, response);  
    return response;  
}
```

Java Collection is not a Cache!

- No Eviction Policies
- No Max Size Limit
(`OutOfMemoryError`)
- No Statistics
- No built-in Cache Loaders
- No Expiration Time
- No Notification Mechanism



Embedded Cache (Java libraries)



```
CacheBuilder.newBuilder()
    .initialCapacity(300)
    .expireAfterAccess(Duration.ofMinutes(10))
    .maximumSize(1000)
    .build();
```

Embedded Cache (Java libraries)



```
CacheBuilder.newBuilder()
    .initialCapacity(300)
    .expireAfterAccess(Duration.ofMinutes(10))
    .maximumSize(1000)
    .build();
```



Caching Application Layer

```
@Service  
public class BookService {  
  
    @Cacheable("books")  
    public String getBookNameByIsbn(String isbn) {  
        return findBookInSlowSource(isbn);  
    }  
  
}
```



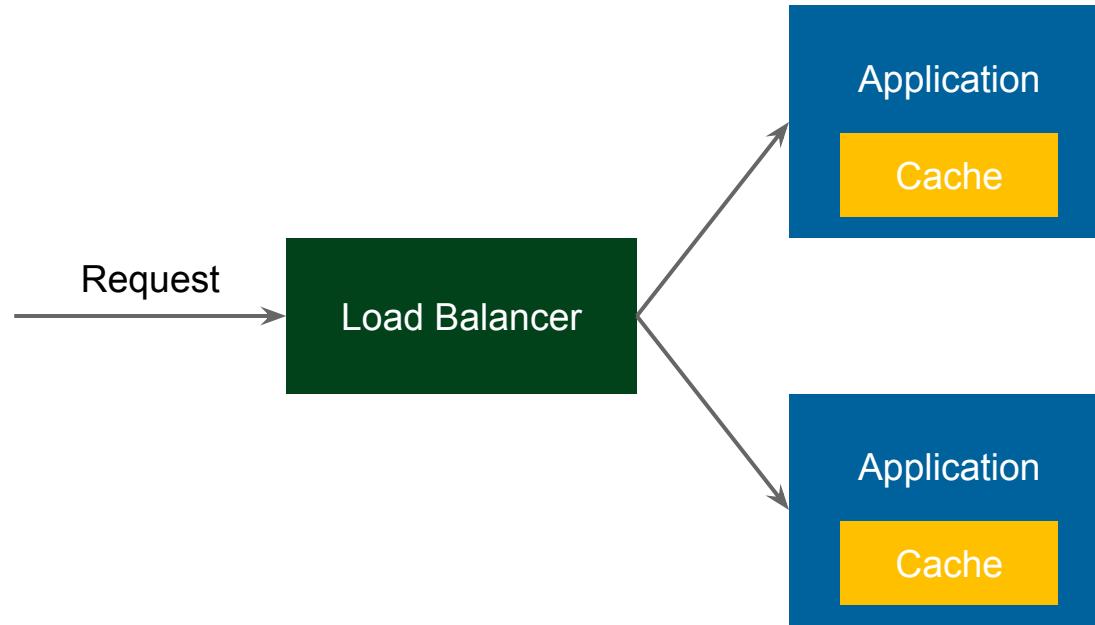
Caching Application Layer

```
@Service  
public class BookService {  
  
    @Cacheable("books")  
    public String getBookNameByIsbn(String isbn) {  
        return findBookInSlowSource(isbn);  
    }  
  
}
```



Be Careful, Spring uses ConcurrentHashMap by default!

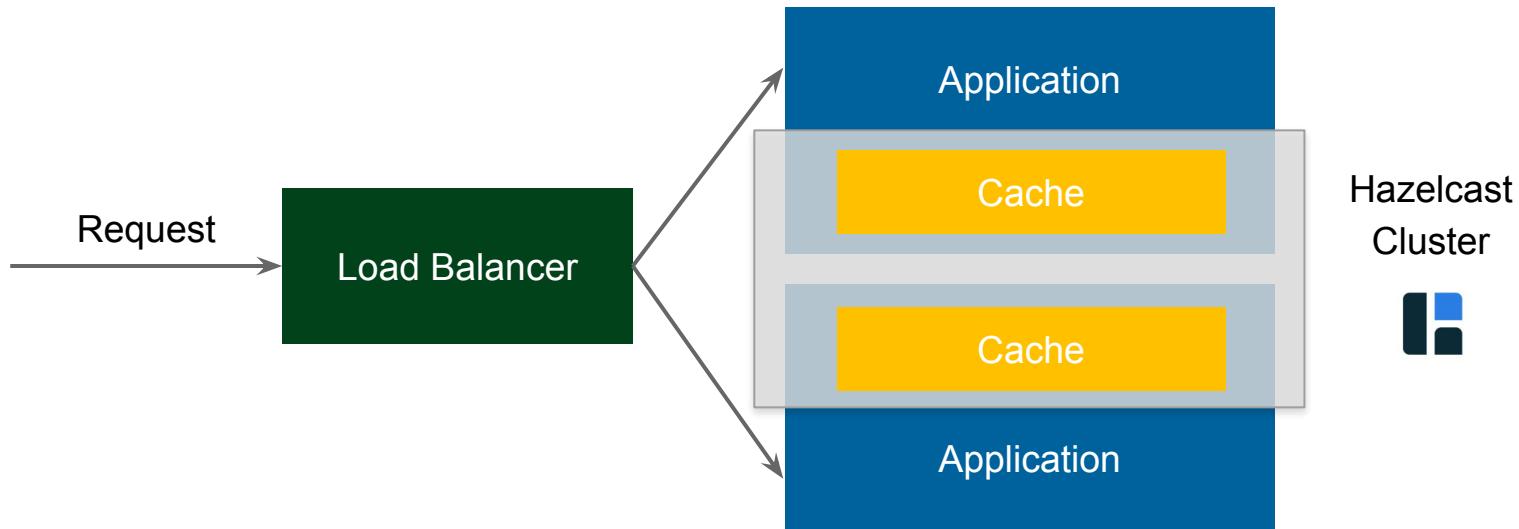
Embedded Cache



The background image shows the Bayon temple complex in Angkor, Cambodia. Several large, multi-tiered stone towers rise from behind a massive tree with thick, light-colored roots. The foreground is filled with fallen stones and debris. A white rectangular box covers the upper portion of the image, containing the text.

1*. Embedded Distributed

Embedded Distributed Cache



Embedded Distributed Cache (Spring with Hazelcast)

```
@Configuration
public class HazelcastConfiguration {

    @Bean
    CacheManager cacheManager() {
        return new HazelcastCacheManager(
            Hazelcast.newHazelcastInstance());
    }

}
```

Hazelcast Discovery Plugins



kubernetes



Google Cloud Platform

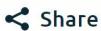


Hazelcast Discovery Plugins

A screenshot of a web browser showing the Hazelcast website. The address bar contains the URL <https://hazelcast.com/blog/how-to-use-embedded-hazelcast-on-kubernetes/>. The page header includes a lock icon, the URL, a star icon, and a search bar. Below the header, there is a navigation bar with links for "Open Source Projects: IMDG | Jet | Training" and a magnifying glass icon. The main navigation menu includes "Products and Services", "Use Cases", "Resources", a three-line menu icon, and a blue "Get Hazelcast" button. The Hazelcast logo is also present.

How to Use Embedded Hazelcast on Kubernetes

Rafal Leszko | February 06, 2019



Share

Hazelcast IMDG is a perfect fit for your (micro)services running on Kubernetes since it can be used in the embedded mode and therefore scale in and out together with your service replicas. This blog post presents a step-by-step description of how to embed Hazelcast into a Spring Boot application and deploy it in the Kubernetes cluster. The source code for this example can be found [here](#).

Hazelcast Discovery Plugins

A screenshot of the Hazelcast website's header. At the top left is the Hazelcast logo. To its right are four main navigation links: "Products and Services", "Use Cases", "Resources", and a menu icon represented by three horizontal lines. On the far right is a blue button labeled "Get Hazelcast". Above the header, there is a browser-style address bar with two entries: "https://hazelcast.com/blog/hazelcast-auto-discovery-with-eureka/" and "https://hazelcast.com/blog/".

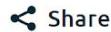


How to Use Hazelcast Auto-Discovery with Eureka

Rafal Leszko | April 24, 2019

How to Use Hazelcast Auto-Discovery with Eureka

Rafal Leszko | February 06, 2019



Hazelcast IMDG supports auto-discovery for many different environments. Since we introduced the [generic discovery SPI](#), a lot of plugins were developed so you can use Hazelcast seamlessly on [Kubernetes](#), [AWS](#), [Azure](#), [GCP](#), and more. Should you need a custom plugin, you are also able to create your own.

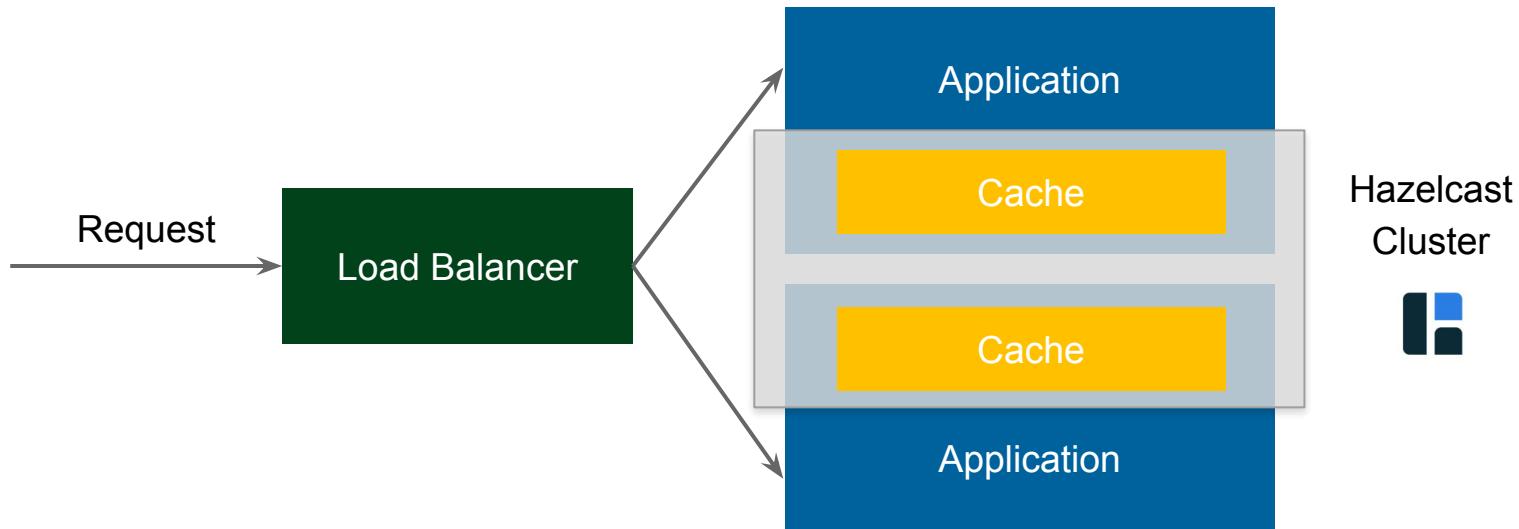
Hazelcast IMDG is a peer-to-peer system, therefore scale in and scale out is very easy.

Hazelcast integrates well with Spring Cloud, you can learn more about it [here](#).

If your infrastructure is not based on any popular Cloud environment, but you still want to take advantage of the dynamic discovery rather than static IP configuration, you can set up your service registry. One of the more popular



Embedded Distributed Cache



Embedded Cache

Pros

- Simple configuration / deployment
- Low-latency data access
- No separate Ops Team needed

Cons

- Not flexible management (scaling, backup)
- Limited to JVM-based applications
- Data collocated with applications

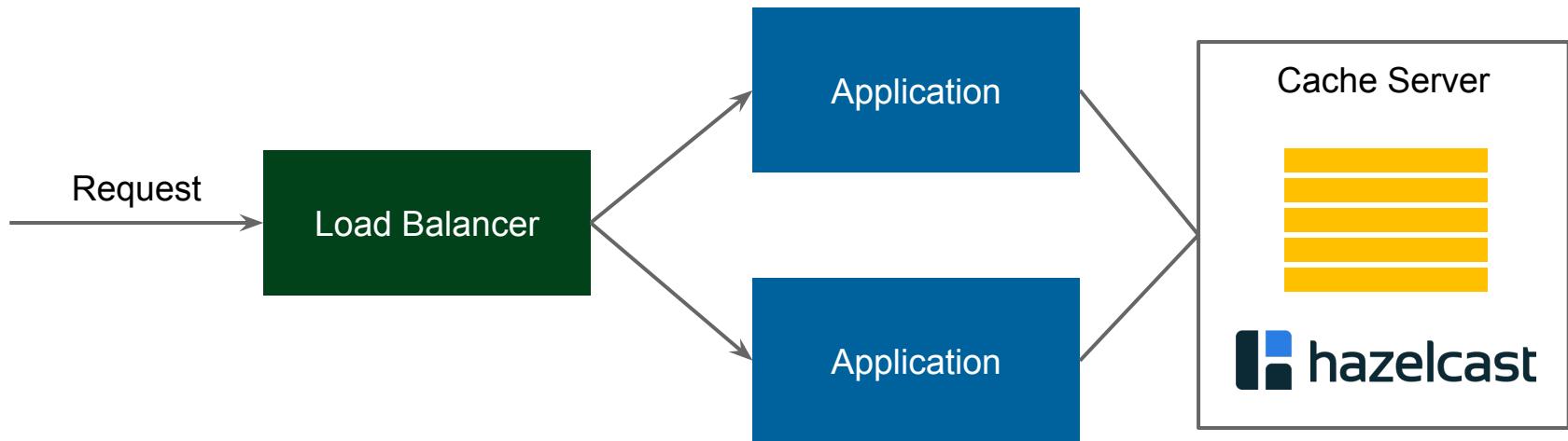
Agenda

- Introduction ✓
- Caching Architectural Patterns
 - Embedded ✓
 - Embedded Distributed ✓
 - Client-Server
 - Cloud
 - Sidecar
 - Reverse Proxy
 - Reverse Proxy Sidecar
- Summary

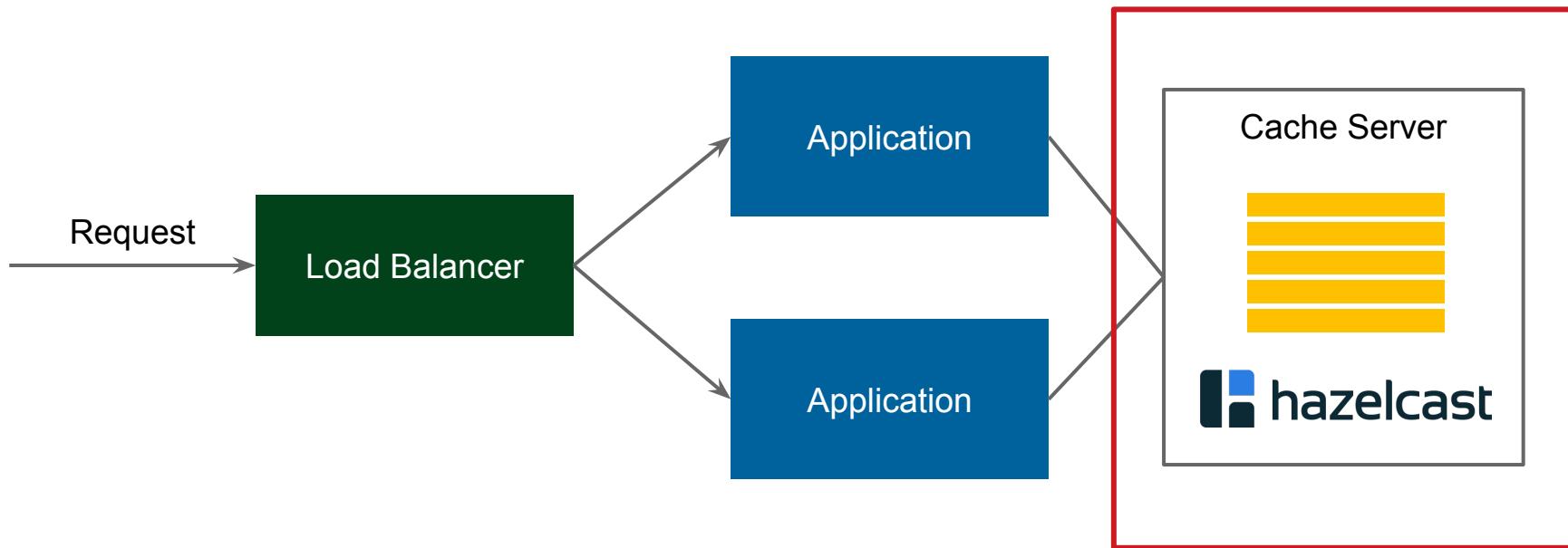
A photograph of a group of people standing on a grassy lawn. In the foreground, several individuals are seen from behind, wearing colorful clothing. One person in a yellow shirt has their arm raised. To the right, a man in maroon robes is holding a camera and taking a picture of the group. The background features large, well-maintained green trees and a dark wooden structure.

2. Client-Server

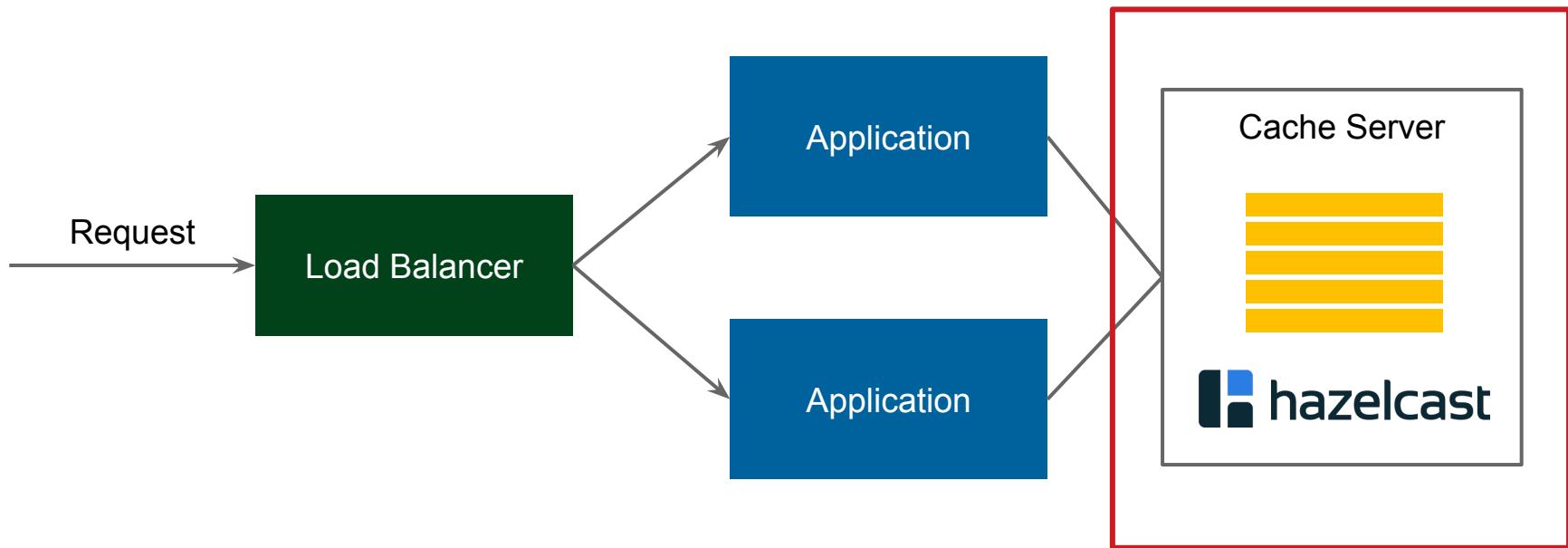
Client-Server Cache



Client-Server Cache



Client-Server Cache



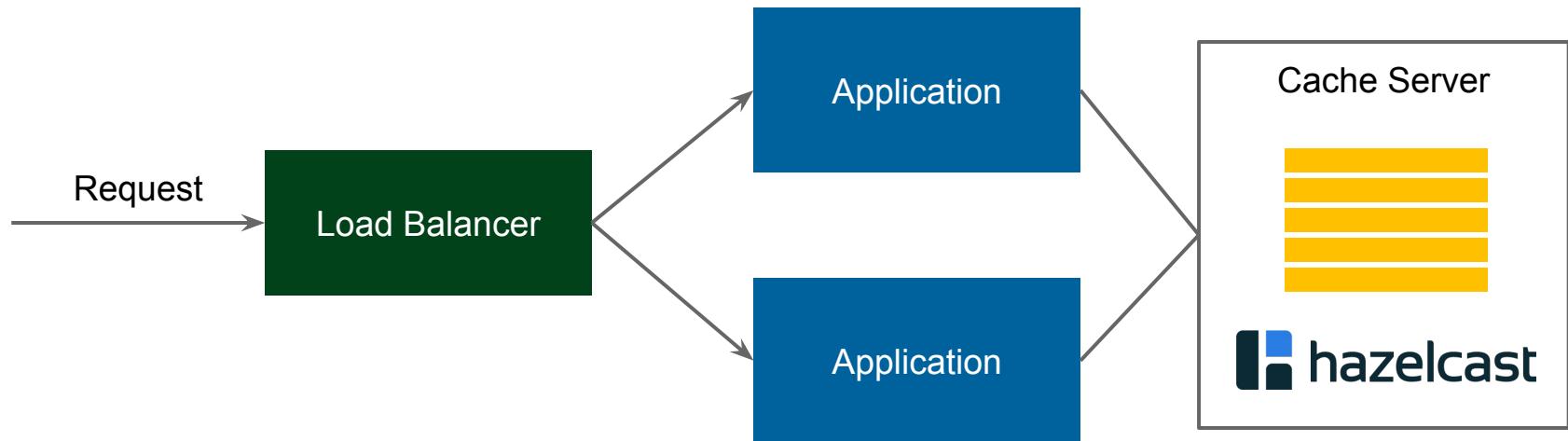
Separate Management:

- backups
- (auto) scaling
- security

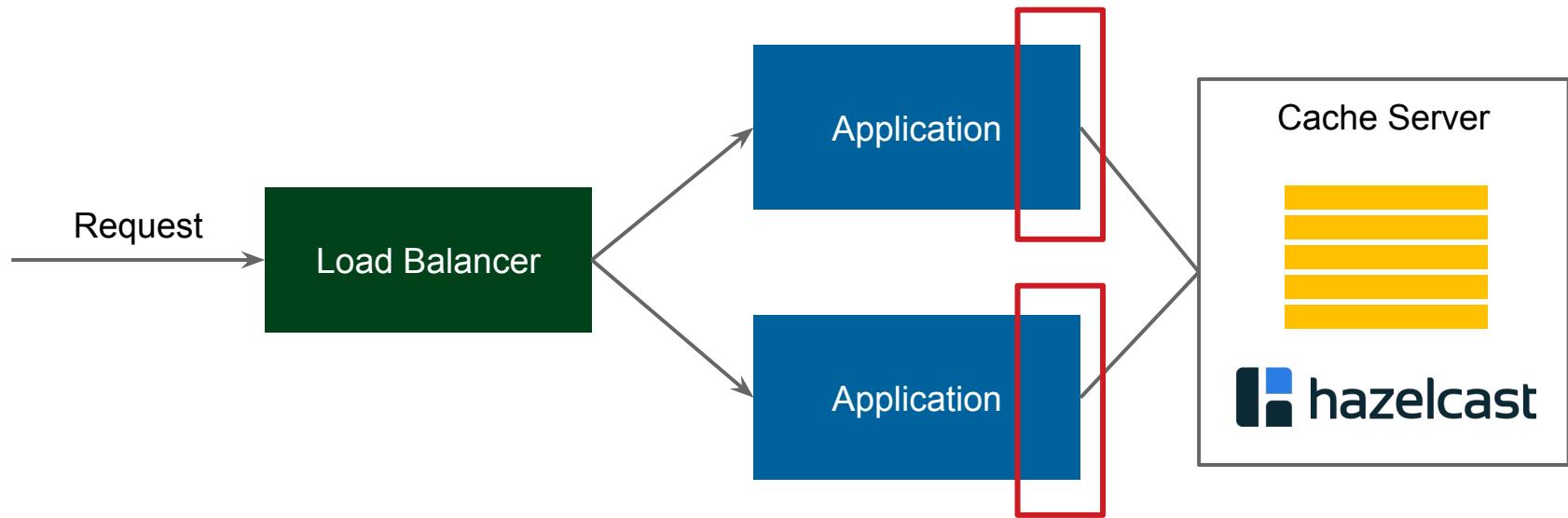


Ops Team

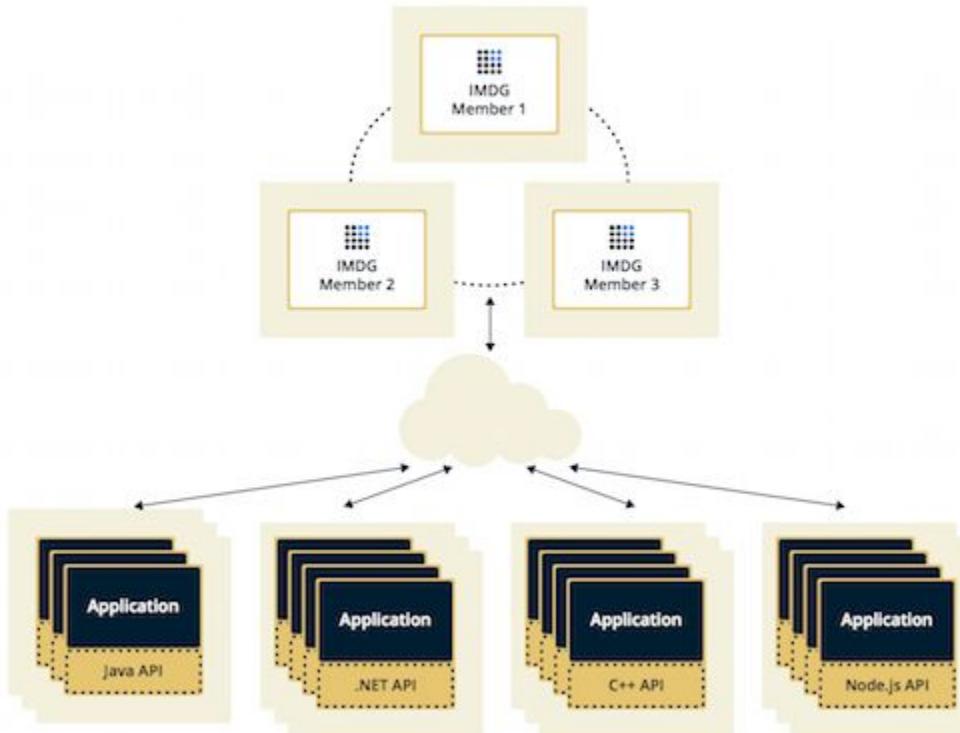
Client-Server Cache



Client-Server Cache



Client-Server Cache



Client-Server Cache



Client-Server Cache

Starting Hazelcast Cache Server (standalone)

```
$ ./start.sh
```

Client-Server Cache

Starting Hazelcast Cache Server (Kubernetes)

```
$ helm install hazelcast/hazelcast
```

Client-Server Cache

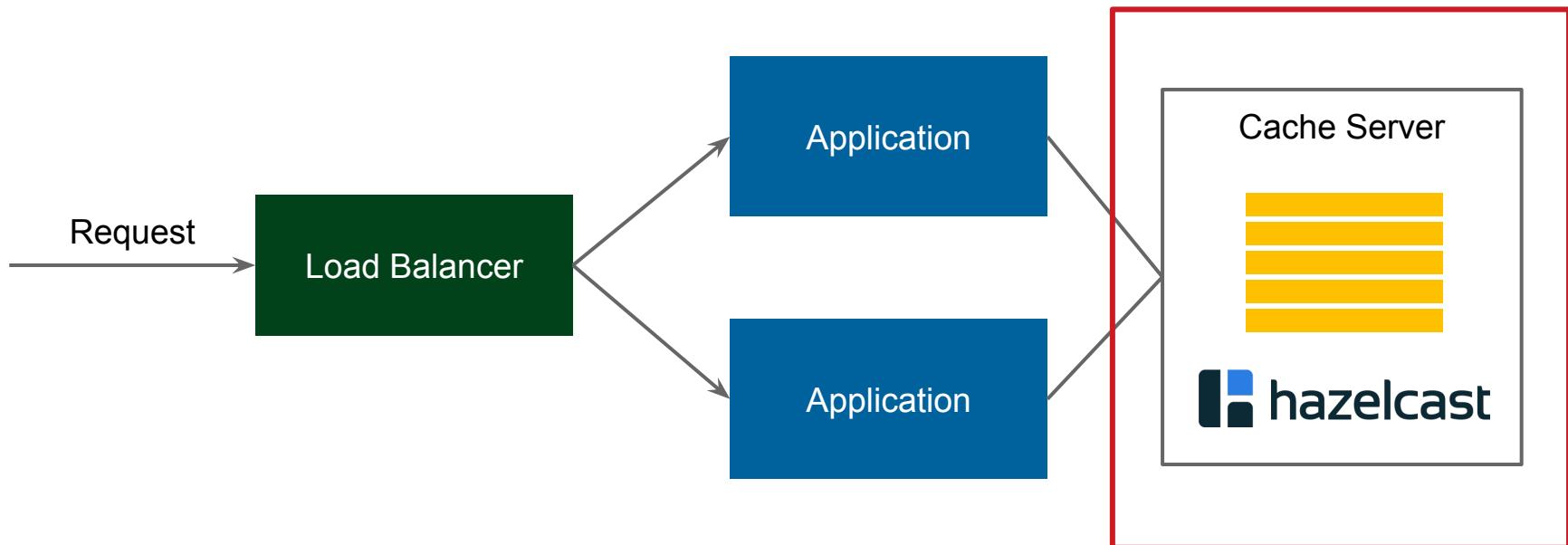
Starting Hazelcast Cache Server (Kubernetes)

```
$ helm install hazelcast/hazelcast
```

Hazelcast Client (Kubernetes):

```
@Configuration
public class HazelcastClientConfiguration {
    @Bean
    CacheManager cacheManager() {
        ClientConfig clientConfig = new ClientConfig();
        clientConfig.getNetworkConfig().getKubernetesConfig()
            .setEnabled(true);
        return new HazelcastCacheManager(HazelcastClient
            .newHazelcastClient(clientConfig));
    }
}
```

Client-Server Cache



Separate Management:

- backups
- (auto) scaling
- security



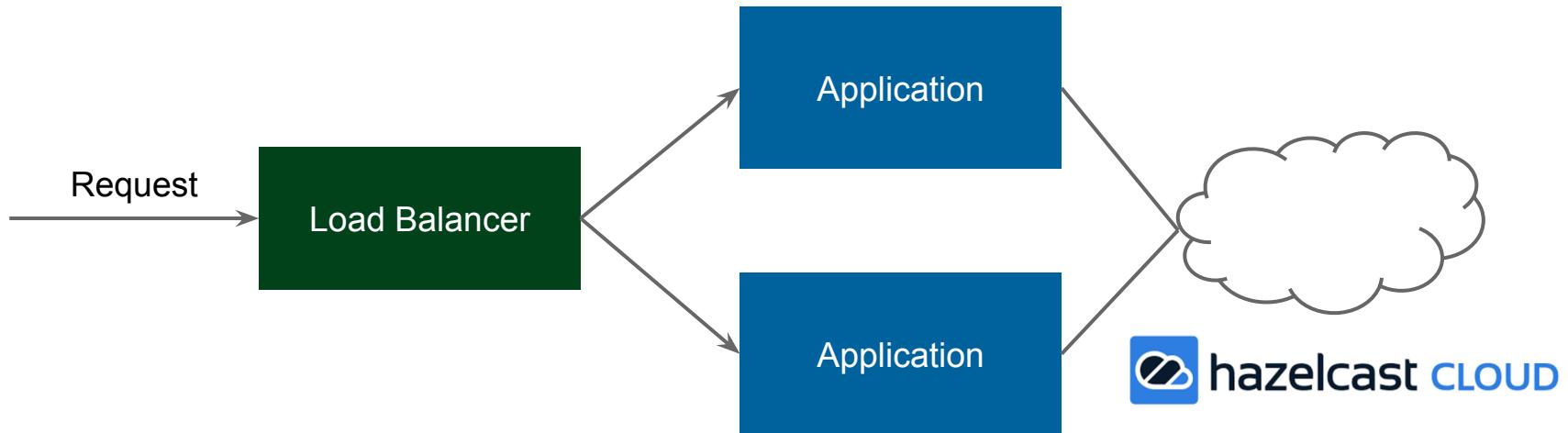
Ops Team



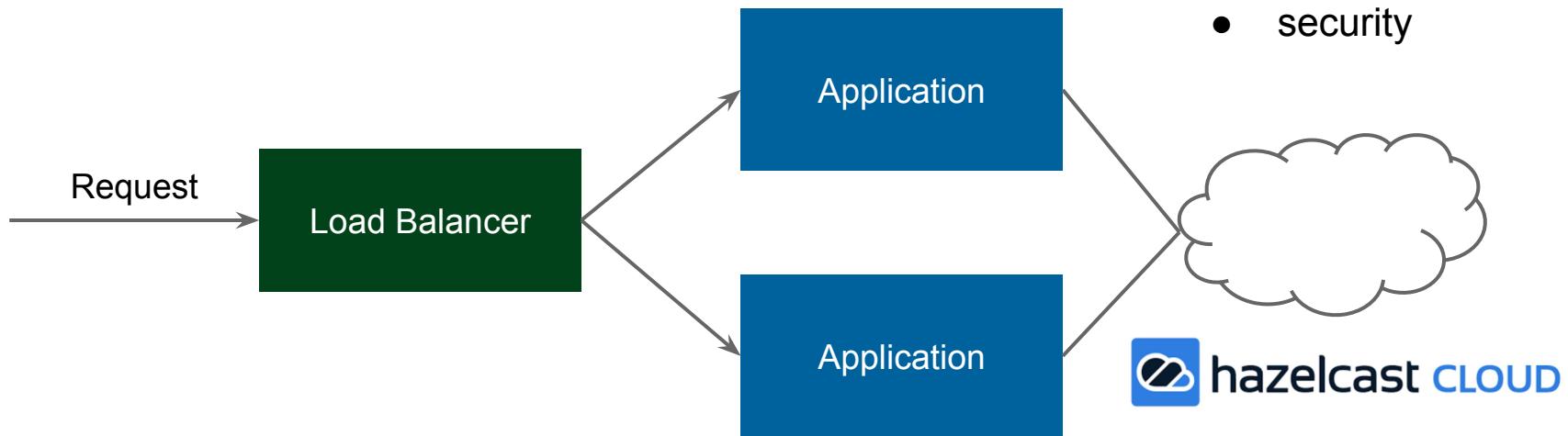
2*. Cloud



Cloud (Cache as a Service)



Cloud (Cache as a Service)



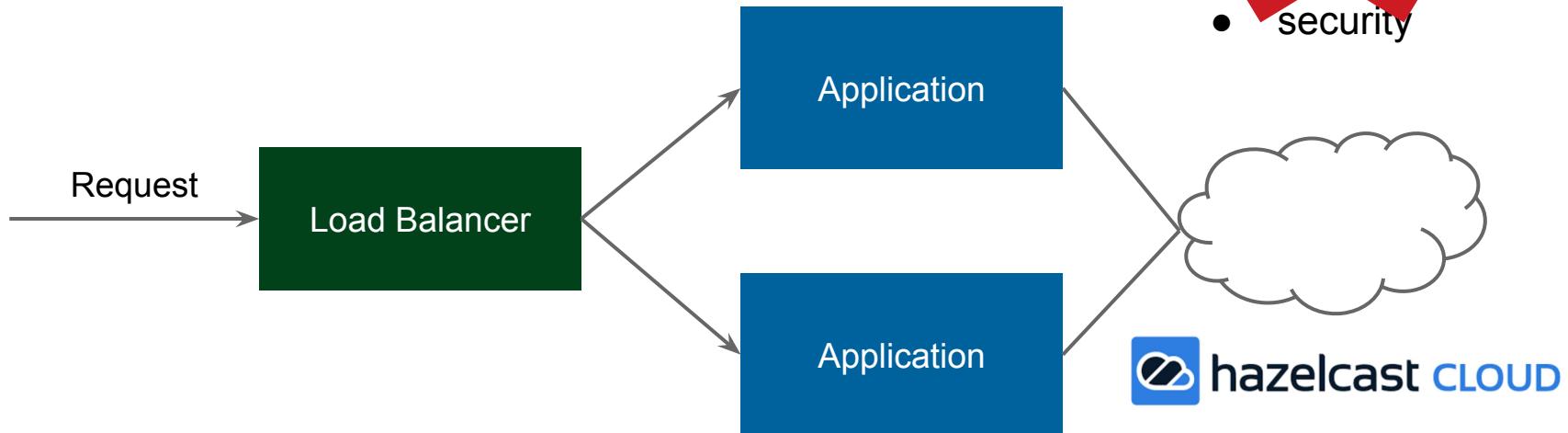
Management:

- backups
- (auto) scaling
- security

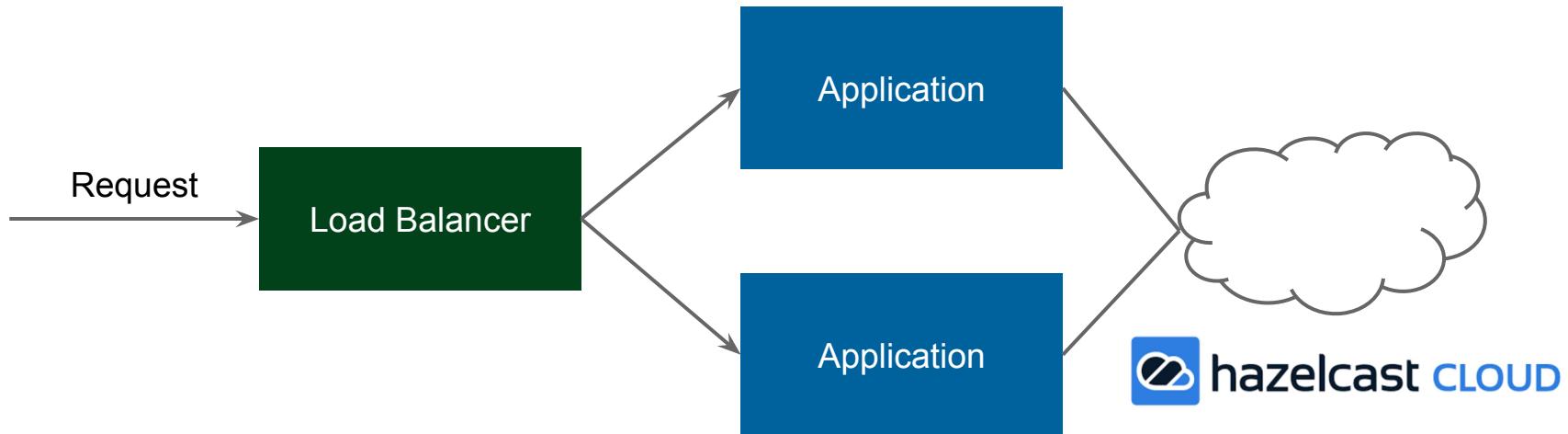


Ops Team

Cloud (Cache as a Service)



Cloud (Cache as a Service)



Cloud (Cache as a Service)

```
@Configuration
public class HazelcastCloudConfiguration {
    @Bean
    CacheManager cacheManager() {
        ClientConfig clientConfig = new ClientConfig();
        clientConfig.getNetworkConfig().getCloudConfig()
            .setEnabled(true)
            .setDiscoveryToken("KSXFDTi5HXPJGR0wRAjLgKe45tvEEhd");
        clientConfig.setGroupConfig(
            new GroupConfig("test-cluster", "b2f984b5dd3314"));
    }
    return new HazelcastCacheManager(
        HazelcastClient.newHazelcastClient(clientConfig));
}
```

DEMO

cloud.hazelcast.com



Client-Server (Cloud) Cache

Pros

- Data separate from applications
- Separate management (scaling, backup)
- Programming-language agnostic

Cons

- Separate Ops effort
- Higher latency
- Server network requires adjustment (same region, same VPC)

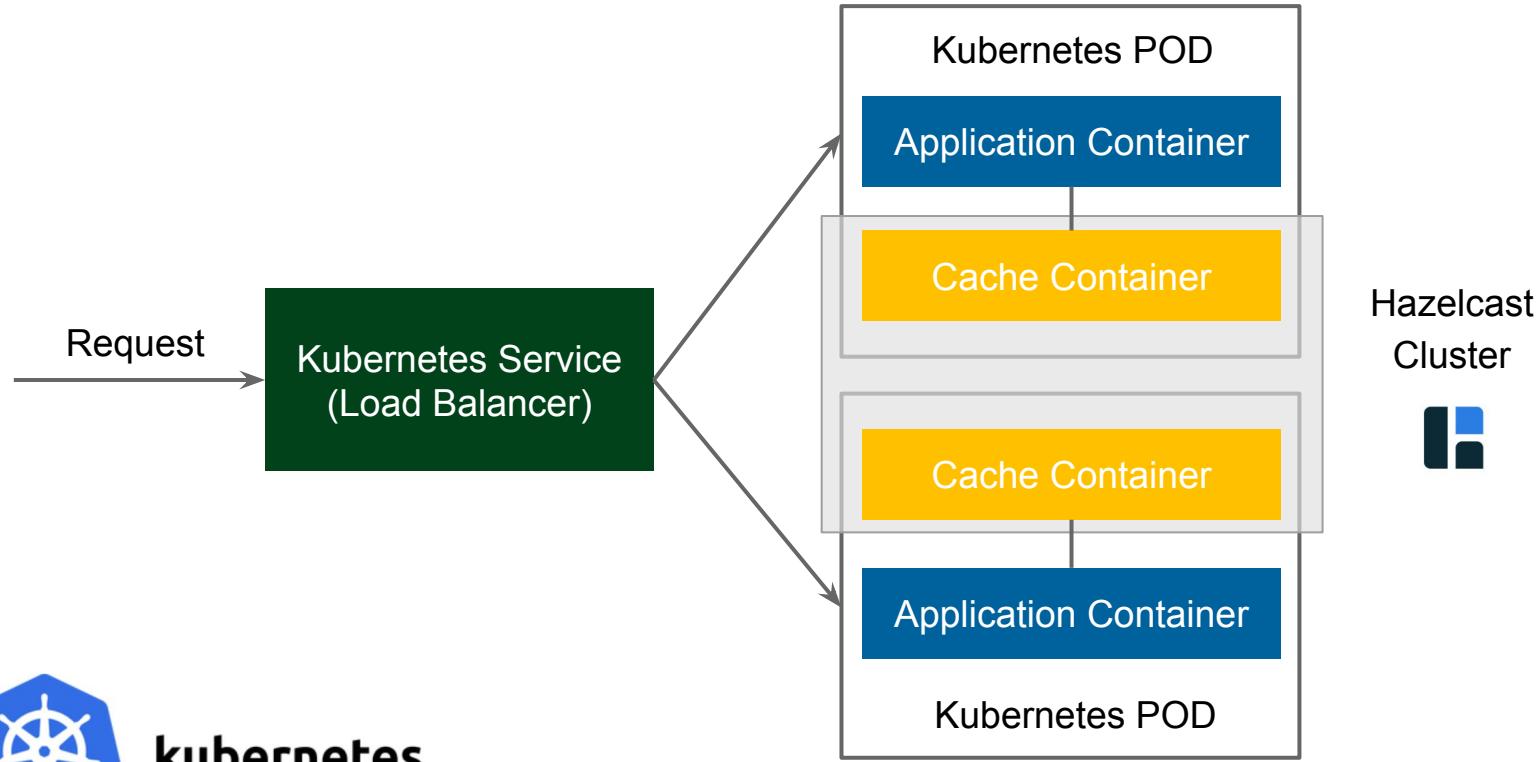
Agenda

- Introduction ✓
- Caching Architectural Patterns
 - Embedded ✓
 - Embedded Distributed ✓
 - Client-Server ✓
 - Cloud ✓
 - Sidecar
 - Reverse Proxy
 - Reverse Proxy Sidecar
- Summary

A dynamic photograph of a sidecar racing team in action. Two drivers wearing red, white, and black Arai helmets are leaning into a turn. The sidecar is white with red and blue accents, featuring various sponsor logos like Velt, Panslaine, and Raimo Puri. The background is blurred green grass, indicating high speed.

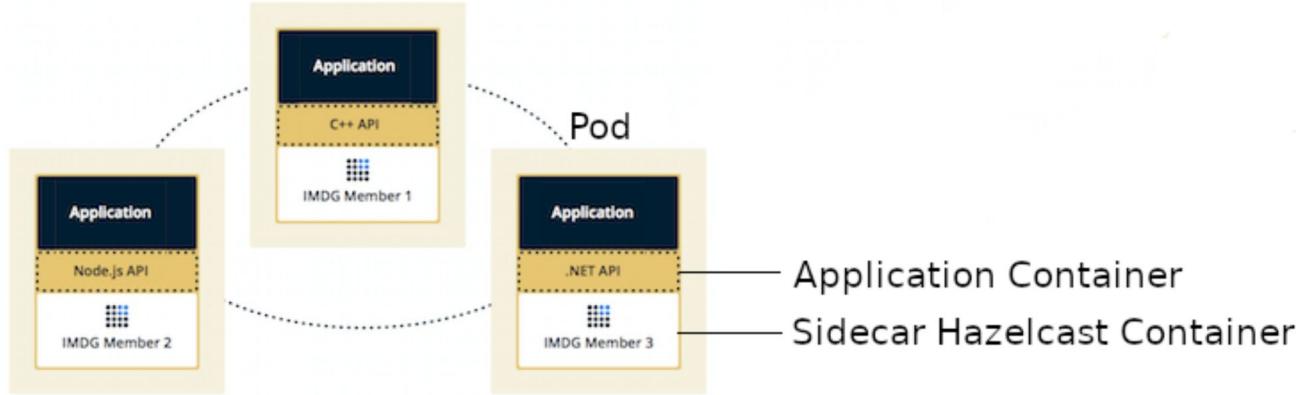
3. Sidecar

Sidecar Cache



kubernetes

Sidecar Cache



Similar to Embedded:

- the same physical machine
- the same resource pool
- scales up and down together
- no discovery needed (always localhost)

Similar to Client-Server:

- different programming language
- uses cache client to connect
- clear isolation between app and cache

Sidecar Cache

```
@Configuration
public class HazelcastSidecarConfiguration {
    @Bean
    CacheManager cacheManager() {
        ClientConfig clientConfig = new ClientConfig();
        clientConfig.getNetworkConfig()
            .addAddress("localhost:5701");
        return new HazelcastCacheManager(HazelcastClient
            .newHazelcastClient(clientConfig));
    }
}
```

Sidecar Cache

```
apiVersion: apps/v1
kind: Deployment
...
spec:
  template:
    spec:
      containers:
        - name: application
          image: leszko/application
        - name: hazelcast
          image: hazelcast/hazelcast
```

Sidecar Cache

Pros

- Simple configuration
- Programming-language agnostic
- Low latency
- Some isolation of data and applications

Cons

- Limited to container-based environments
- Not flexible management (scaling, backup)
- Data collocated with application PODs

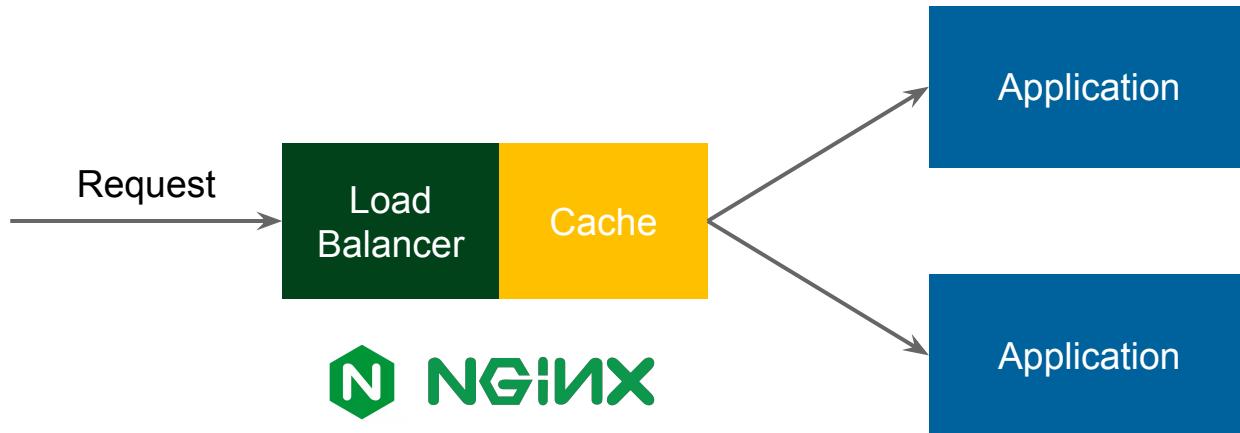
Agenda

- Introduction ✓
- Caching Architectural Patterns
 - Embedded ✓
 - Embedded Distributed ✓
 - Client-Server ✓
 - Cloud ✓
 - Sidecar ✓
 - Reverse Proxy
 - Reverse Proxy Sidecar
- Summary

4. Reverse Proxy



Reverse Proxy Cache



Reverse Proxy Cache



```
http {  
    ...  
    proxy_cache_path /data/nginx/cache  
        keys_zone=one:10m;  
    ...  
}
```

NGINX Reverse Proxy Cache Issues

- Only for HTTP
- Not distributed
- No High Availability
- Data stored on the disk

NGINX Reverse Proxy Cache Issues

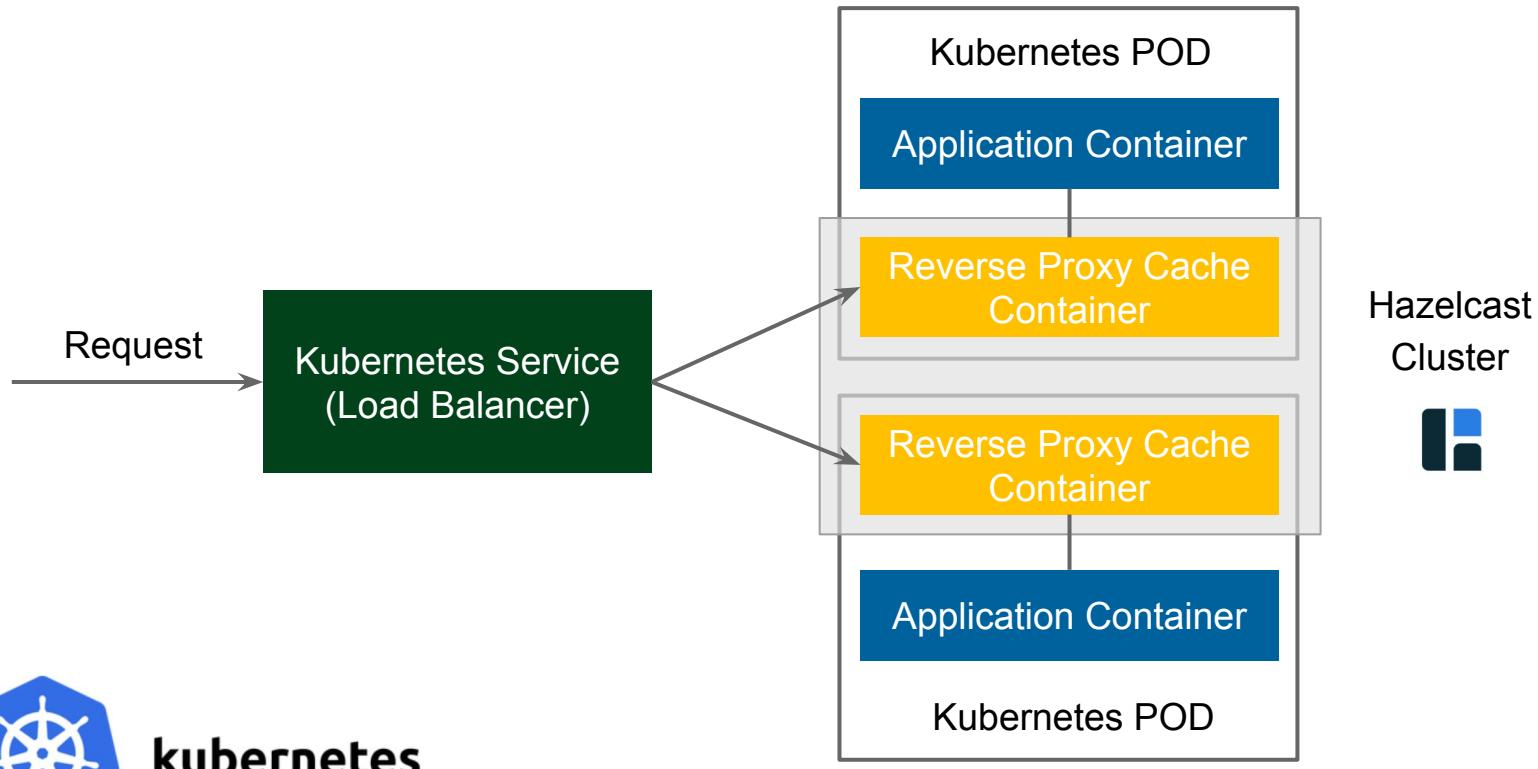
- Only for HTTP
- Not distributed
- No High Availability
- Data stored on the disk



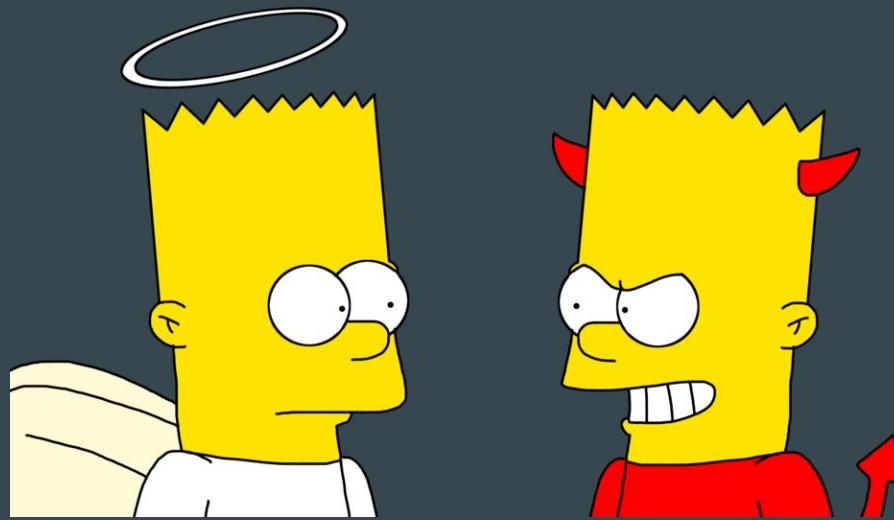


4*. Reverse Proxy Sidecar

Reverse Proxy Sidecar Cache



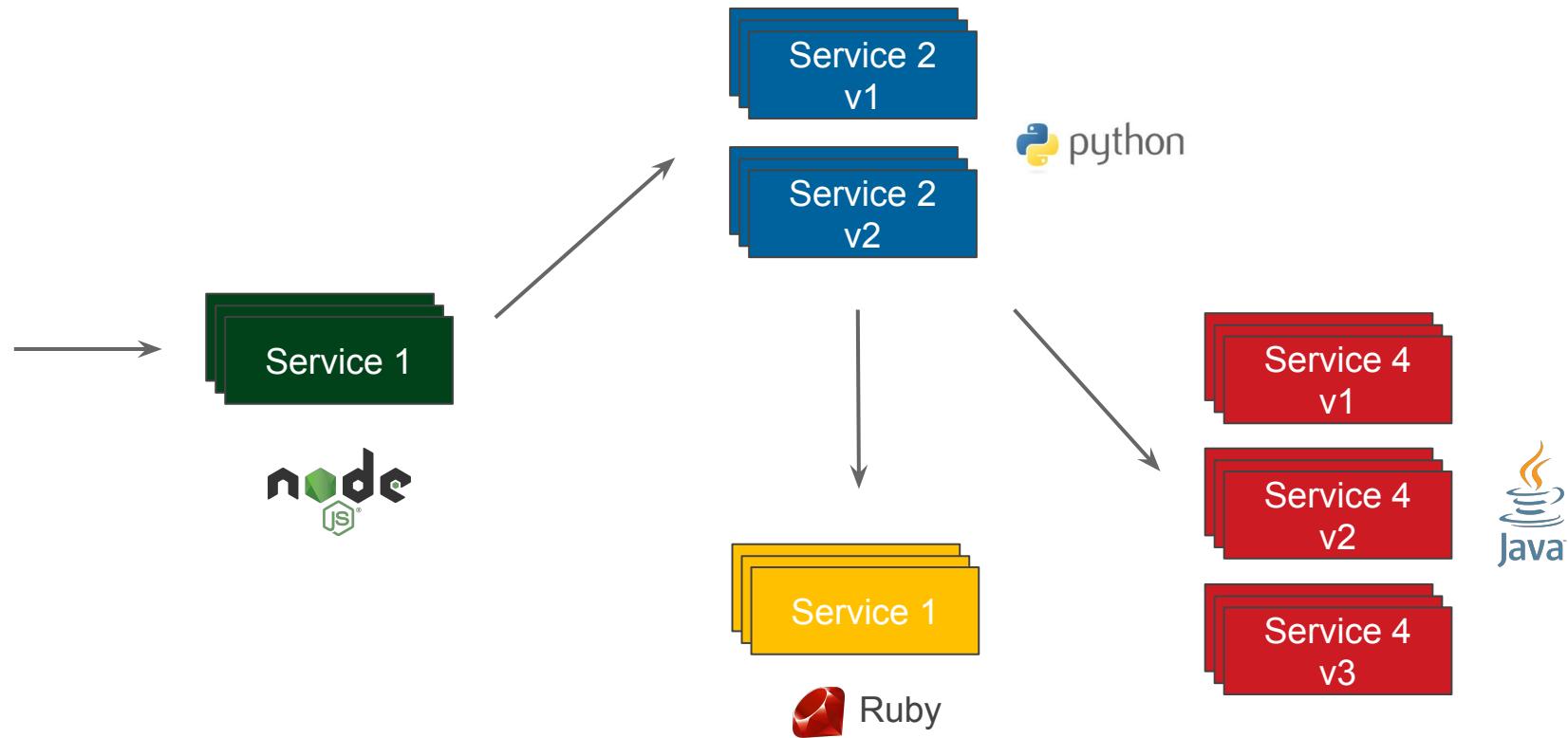
kubernetes



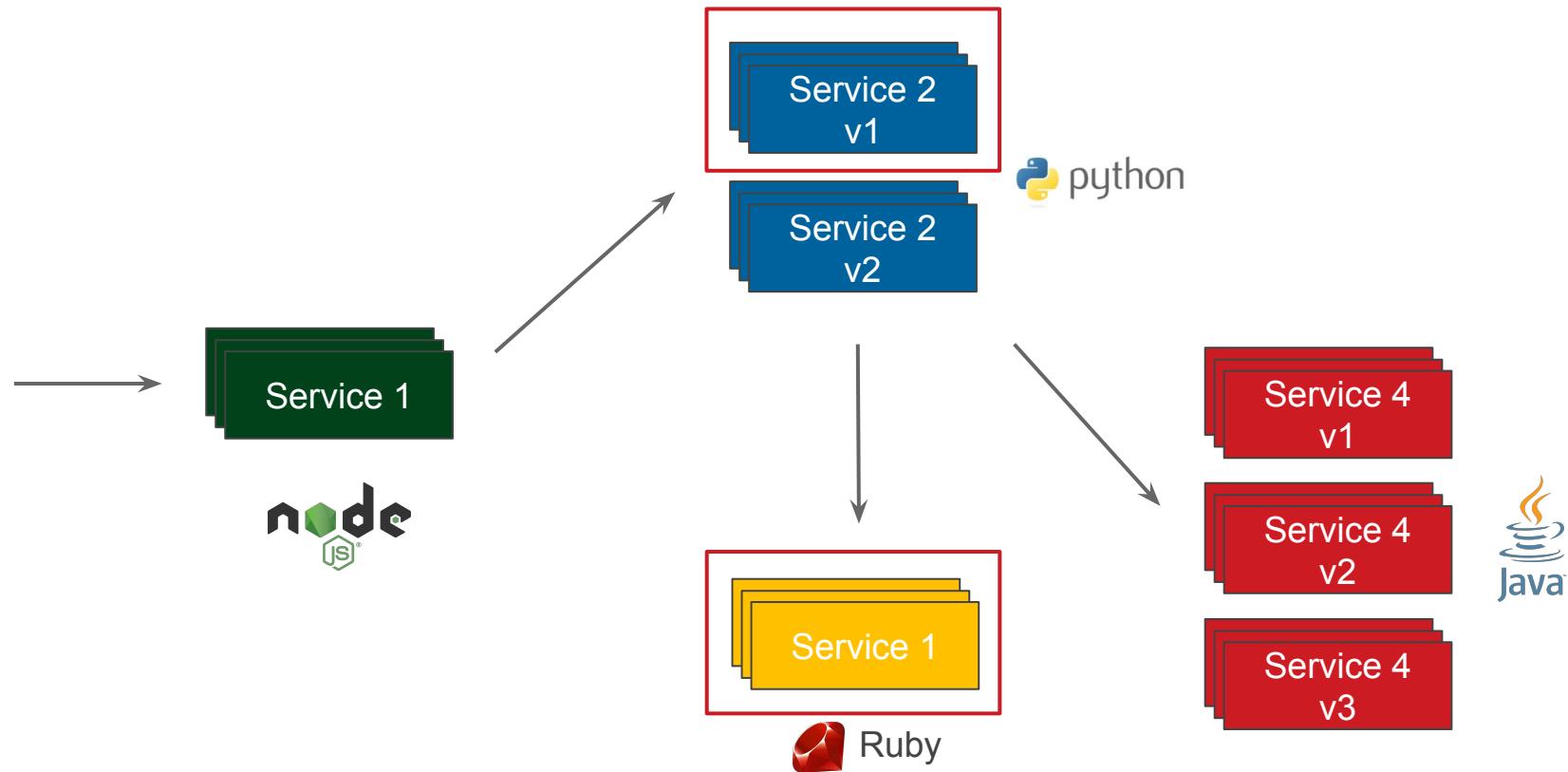


Good

Reverse Proxy Sidecar Cache



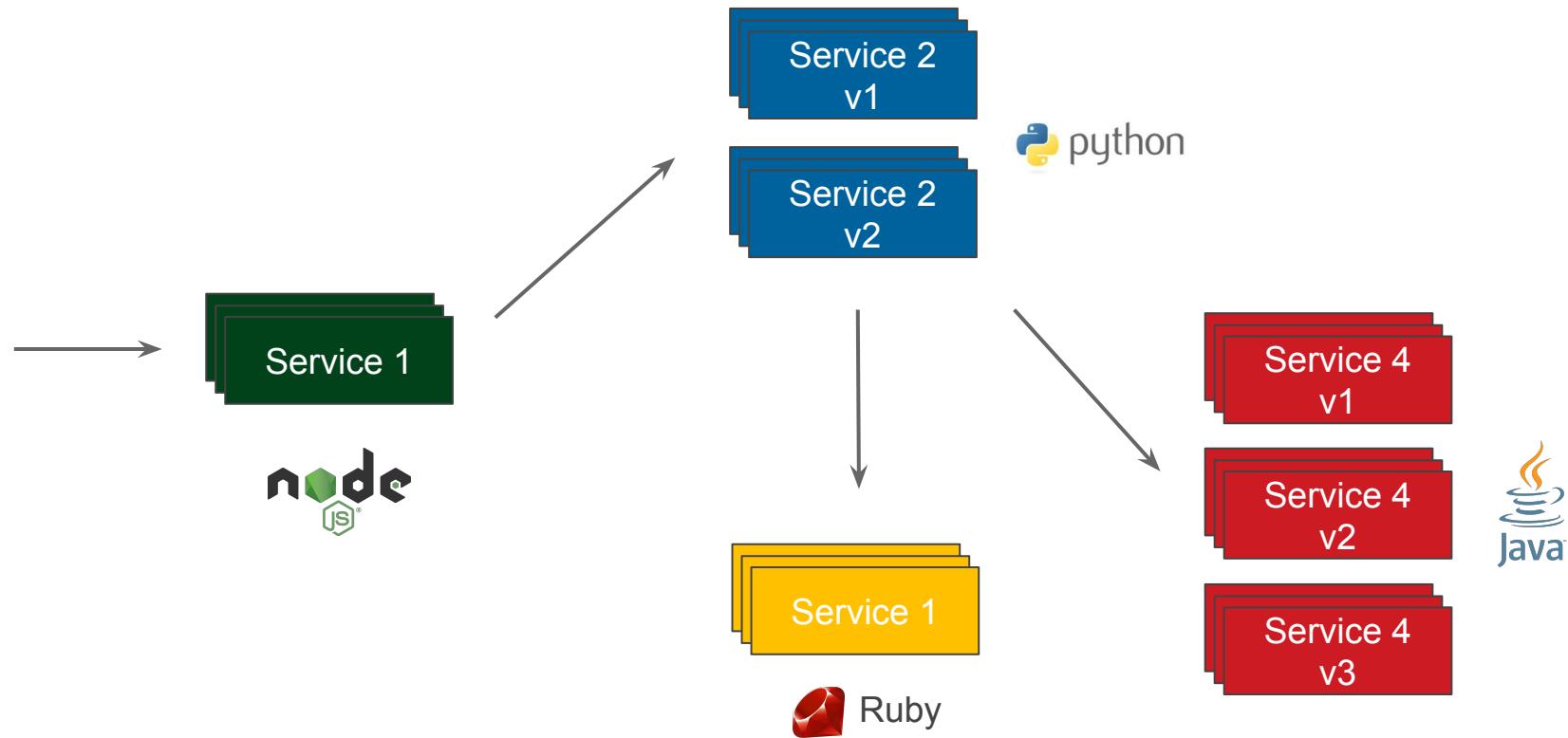
Reverse Proxy Sidecar Cache

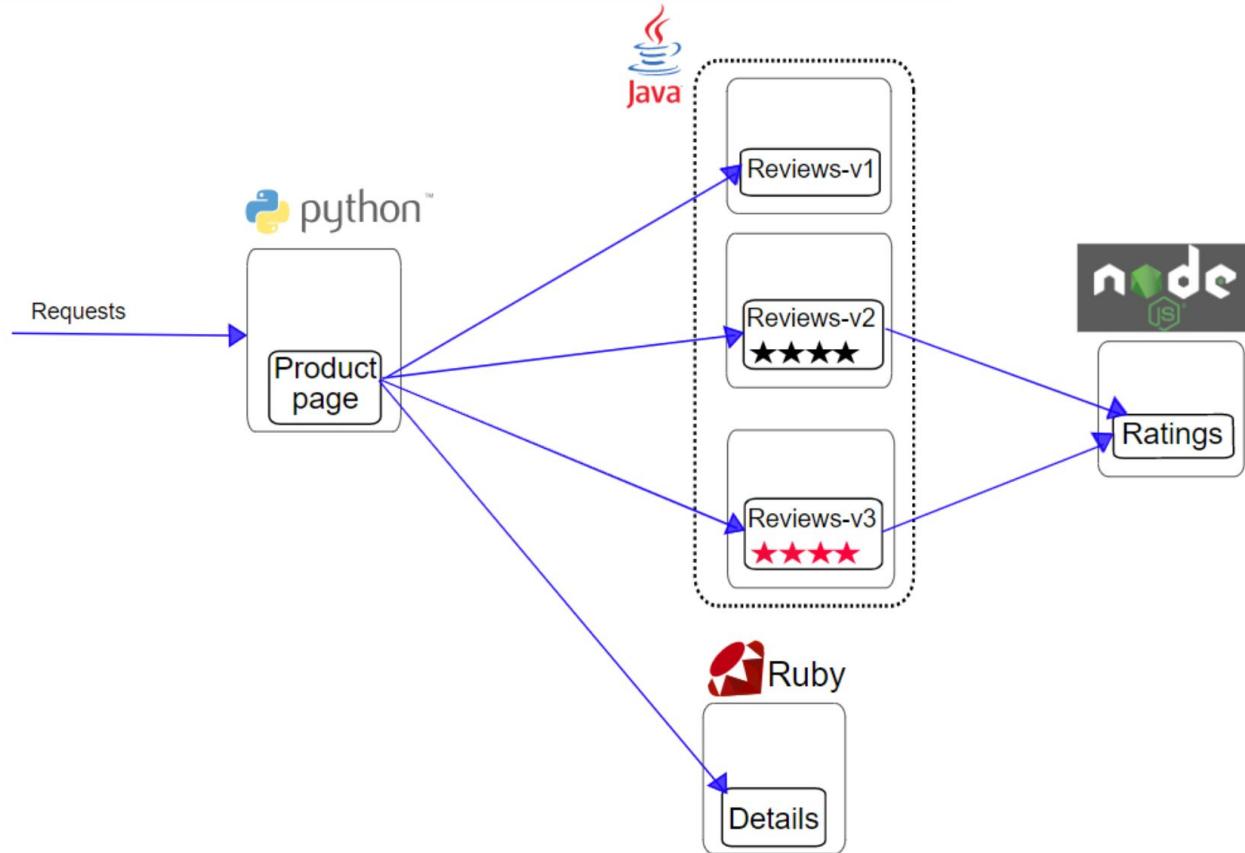


Reverse Proxy Sidecar Cache

```
apiVersion: apps/v1
kind: Deployment
...
spec:
  template:
    spec:
      initContainers:
        - name: init-networking
          image: leszko/init-networking
      containers:
        - name: caching-proxy
          image: leszko/caching-proxy
        - name: application
          image: leszko/application
```

Reverse Proxy Sidecar Cache





Reverse Proxy Sidecar Cache (Istio)

 [envoyproxy / envoy](#) Watch ▾ 509 Star

[Code](#) Issues 438 [Pull requests 51](#) [Projects 1](#) [Insights](#)

Support http caching #868

Open tschroed opened this issue on May 1, 2017 · 10 comments

 **tschroed** commented on May 1, 2017 Contributor + 😊 ...

As a generic http proxy, it would be useful for Envoy to support http caching. It seems like this could probably implemented as a filter.

 21

Bad



What are some best practices for caching in a typical web app?

This question previously had details. They are now in a comment.

Answer

Follow · 48

Request

1



...

4 Answers



Kellan Elliott-McCREA

Answered Sep 4, 2010



The hardest part of caching is cache invalidation. If you're a content driven site, then your job is trivial. If you are building anything resembling social software caching involves a series of complex trade offs.

Reverse Proxy Cache

Application Cache:

```
@CacheEvict(value = "someValue", allEntries = true)  
public void evictAllCacheValues() { }
```

Reverse Proxy Cache

Application Cache:

```
@CacheEvict(value = "someValue", allEntries = true)  
public void evictAllCacheValues() { }
```

Proxy Cache:

```
http {  
    ...  
    location / {  
        add_header Cache-Control public;  
        expires 86400;  
        etag on;  
    }  
}
```

Reverse Proxy (Sidecar) Cache

Pros

- Configuration-based (no need to change applications)
- Programming-language agnostic
- Consistent with containers and microservice world

Cons

- Difficult cache invalidation
- No mature solutions yet
- Protocol-based (e.g. works only with HTTP)

Agenda

- Introduction ✓
- Caching Architectural Patterns
 - Embedded ✓
 - Embedded Distributed ✓
 - Client-Server ✓
 - Cloud ✓
 - Sidecar ✓
 - Reverse Proxy ✓
 - Reverse Proxy Sidecar ✓
- Summary



Summary

application-aware?

application-aware?

no

containers?

application-aware?

no

containers?

no

Reverse Proxy

application-aware?

no

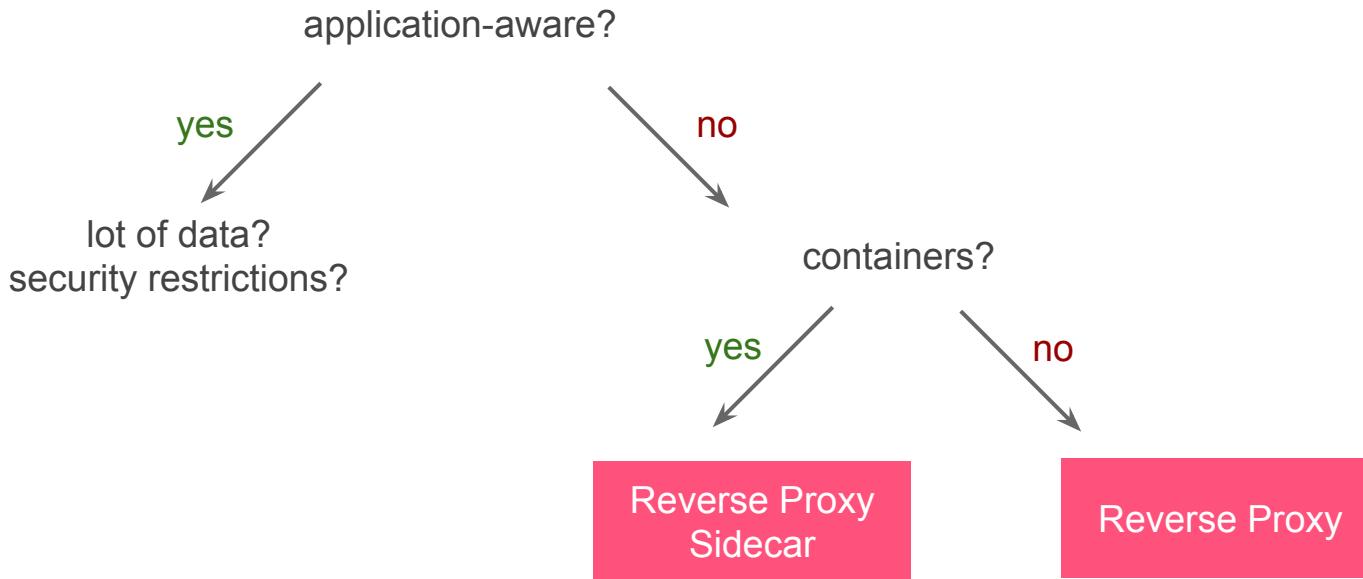
containers?

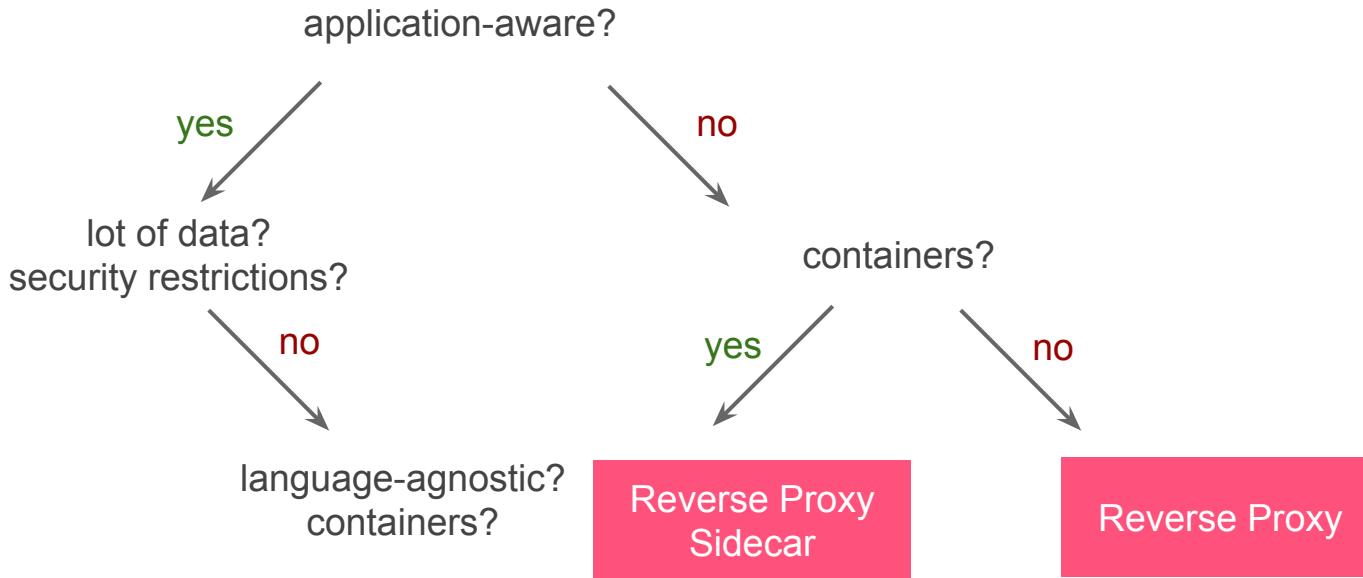
yes

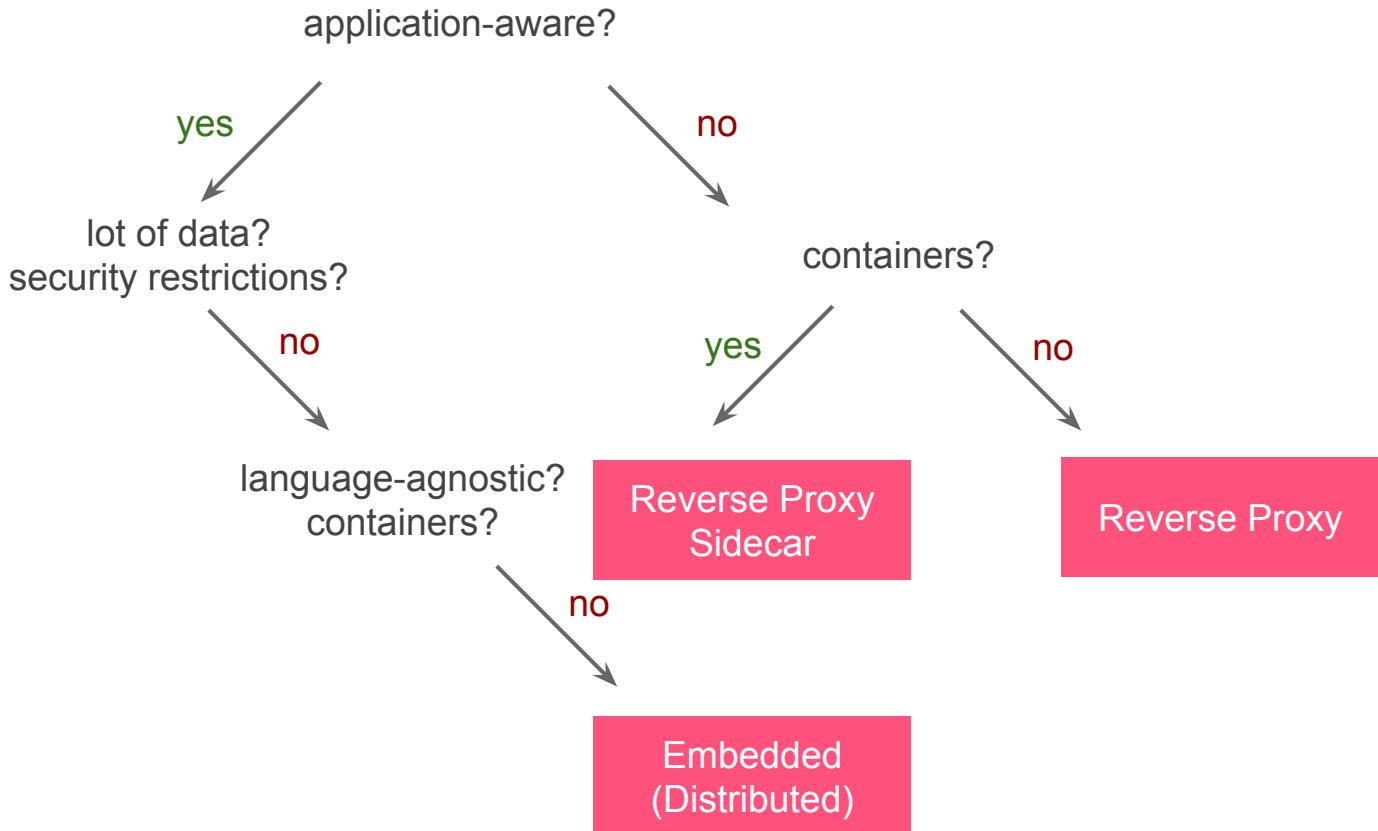
no

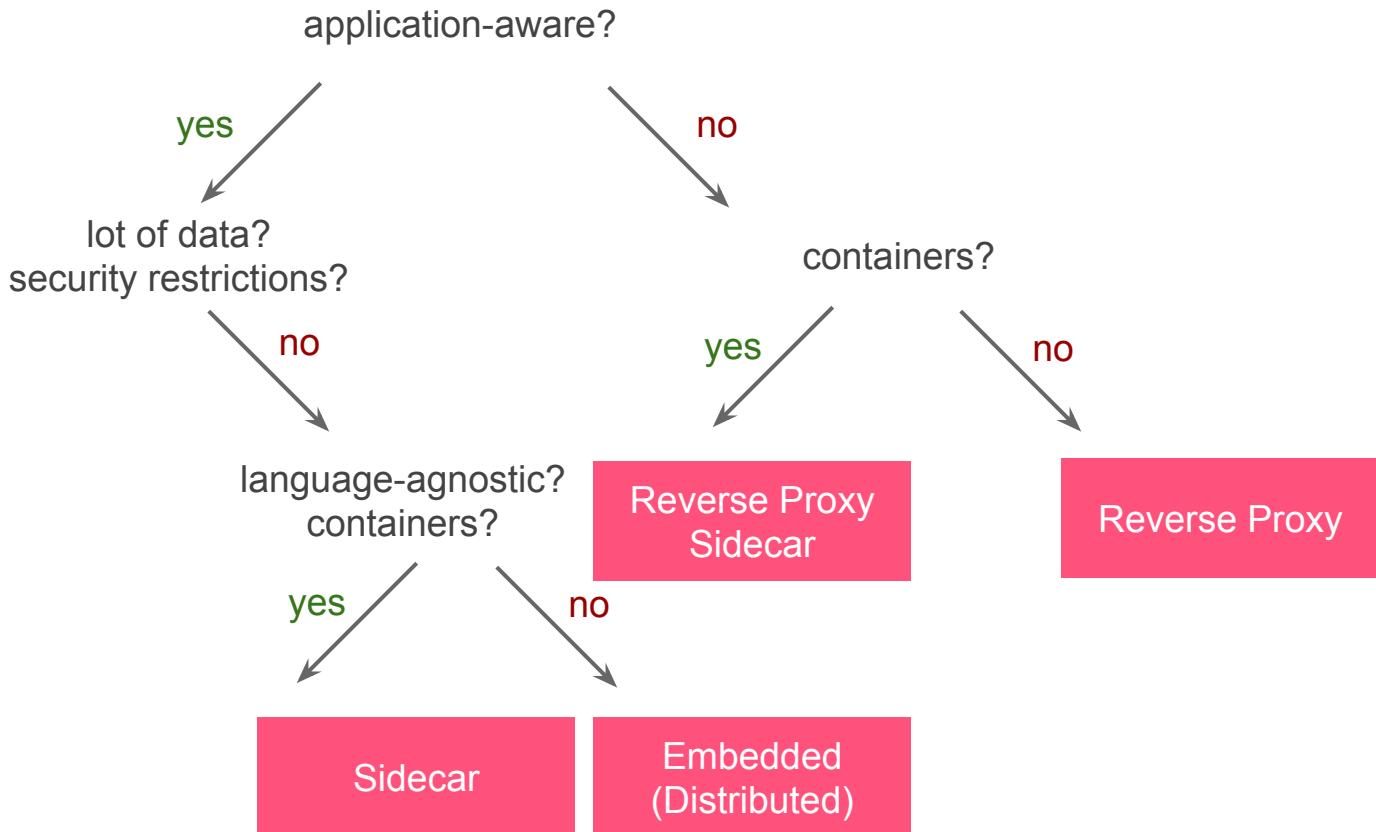
Reverse Proxy
Sidecar

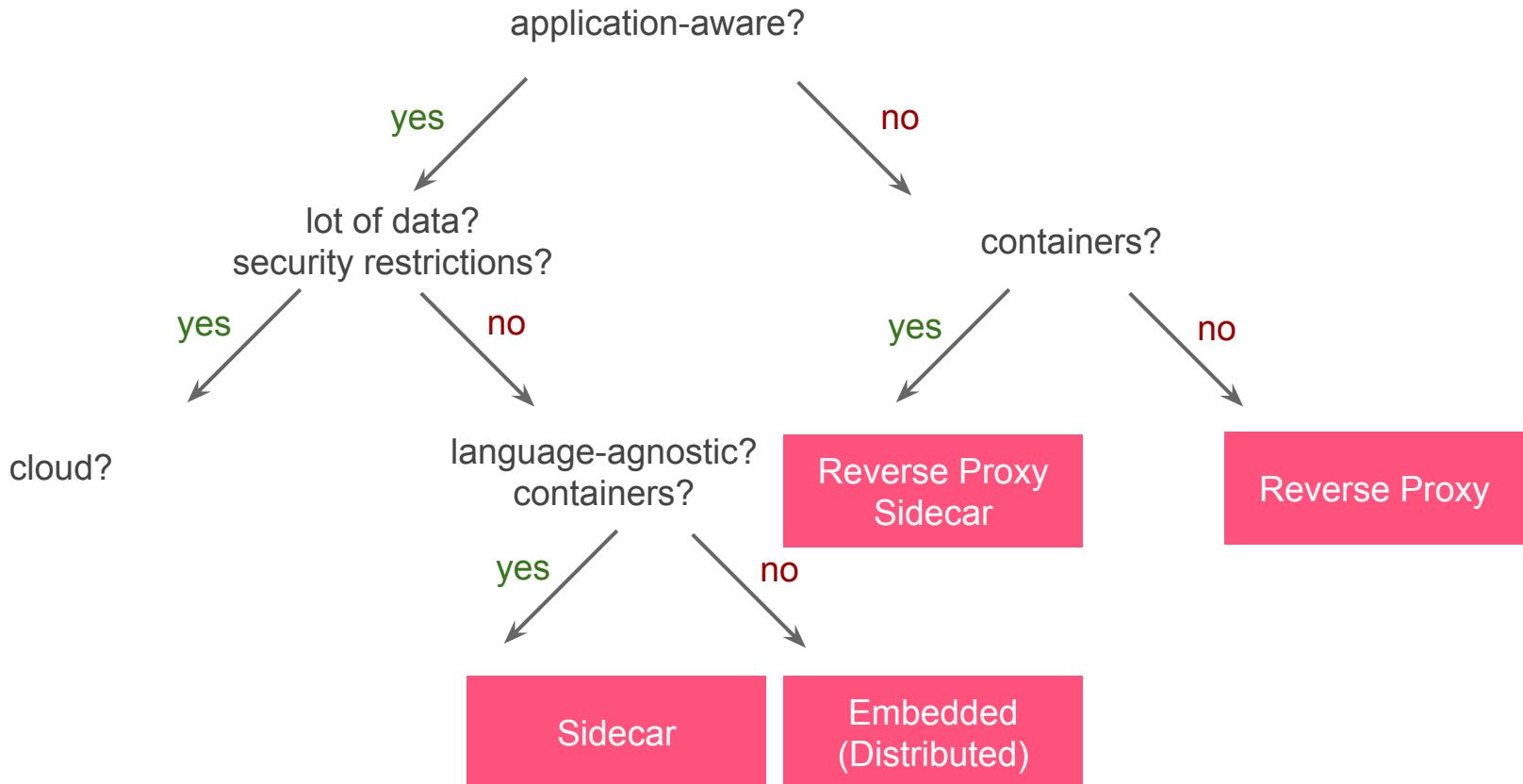
Reverse Proxy

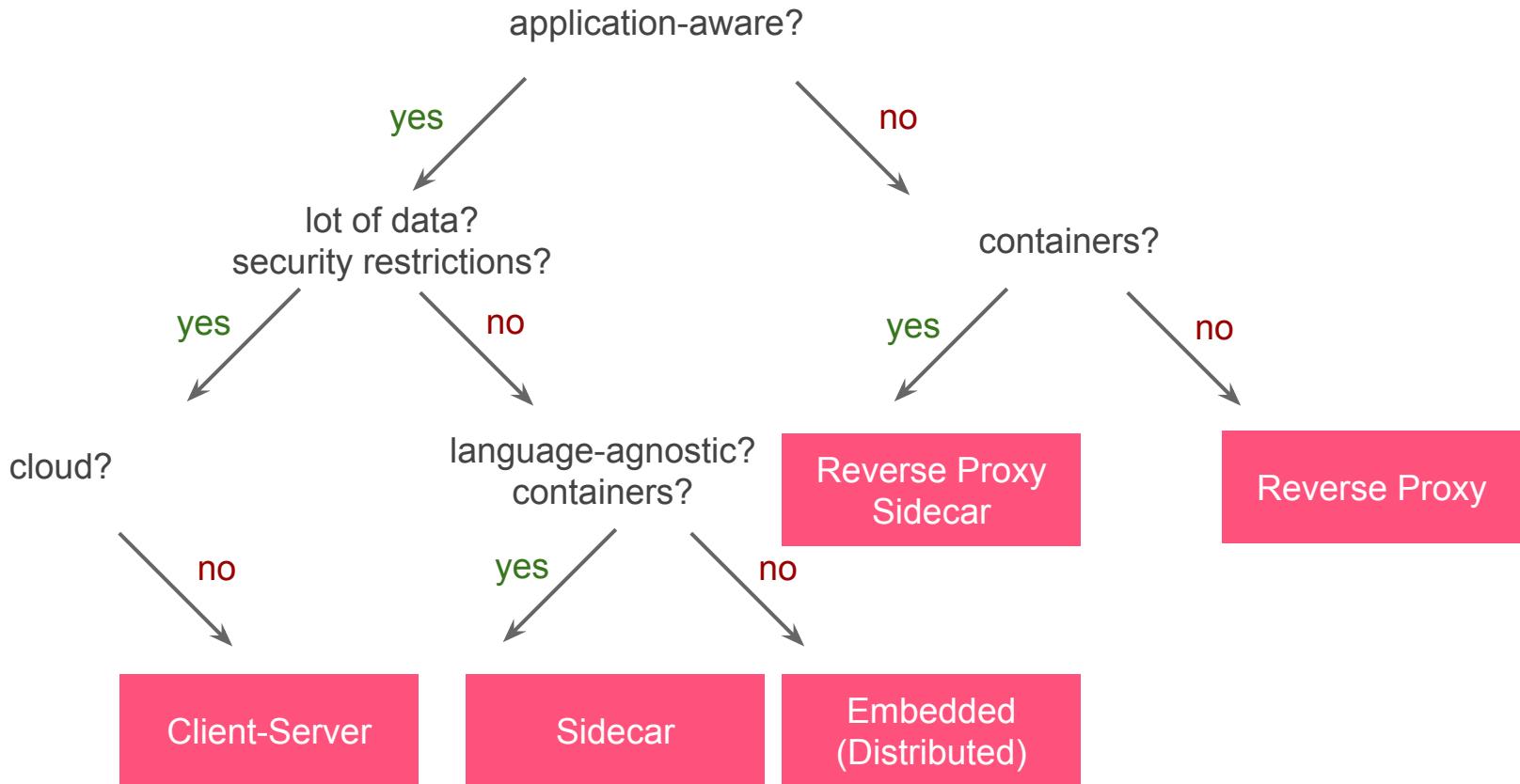


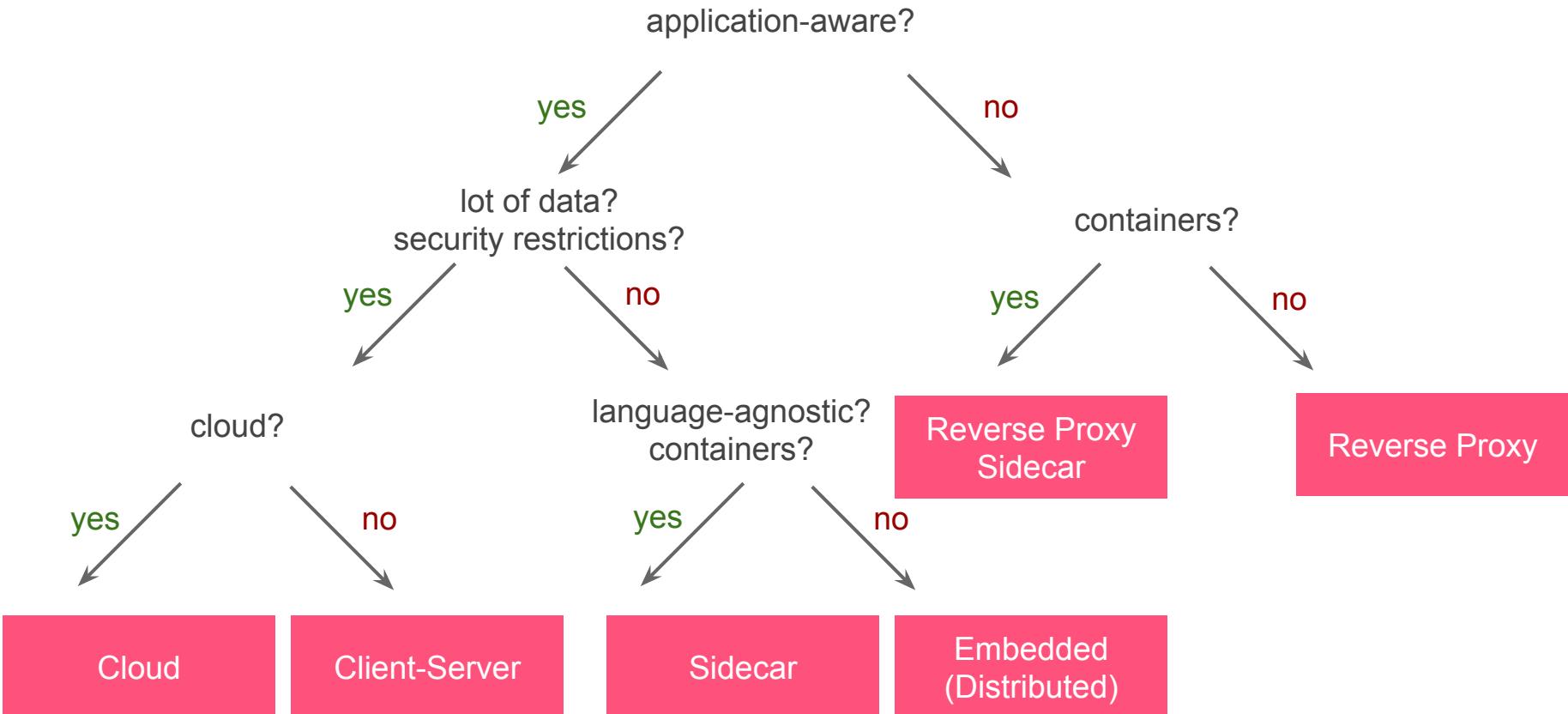












Resources

- Hazelcast Sidecar Container Pattern:
<https://hazelcast.com/blog/hazelcast-sidecar-container-pattern/>
- Hazelcast Reverse Proxy Sidecar Caching Prototype:
<https://github.com/leszko/caching-injector>
- Caching Best Practices:
<https://vladmihalcea.com/caching-best-practices/>
- NGINX HTTP Reverse Proxy Caching:
<https://www.nginx.com/resources/videos/best-practices-for-caching>

L

Thank You!

Rafał Leszko



@RafalLeszko