International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

Phased Rollout Configuration: A Comprehensive Approach for Feature Releases in Software Systems

Mahidhar Mullapudi

Abstract: ***Effective feature rollout strategies [1] are crucial for ensuring a smooth and controlled deployment of new functionalities in software systems [2]. This paper introduces a novel approach to phased rollouts using an advanced configuration library. The library facilitates the configuration of feature rollouts based on the type of entity, allowing for fine - grained control and flexibility. This paper explores the key principles [3], design considerations, and implementation details of the proposed rollout configuration library. Through practical examples, we demonstrate how this approach enhances the reliability, manageability, and customization of feature deployments in complex software environments. Additionally, the paper outlines essential criteria for deploying this library in large - scale projects [4][5][6].***

Keywords: Modern Distributed Applications, Rollout Configuration Library, Phased Feature Rollout

1. Introduction

In the dynamic landscape of software development, large - scale projects present unique challenges in managing feature deployments. The ability to deploy new features seamlessly and with minimal impact on existing systems becomes increasingly complex as the size of the project grows. Traditional rollout strategies often fall short in addressing the specific requirements of large - scale projects, necessitating a more sophisticated approach [7][8].

This paper not only introduces a groundbreaking rollout configuration library but also outlines crucial criteria to consider when implementing this library in large - scale projects. We delve into the core principles of the library's design, emphasizing its scalability, performance, and adaptability to diverse entities within expansive software ecosystems [9].

Criteria for Large - Scale Projects:

1) Scalability:
The library should be designed to scale seamlessly with the size of the project, supporting the rollout of features across a multitude of entities without compromising performance [9][10].

2) Performance Optimization:
Considerations for optimizing performance during feature rollout, ensuring minimal impact on system resources and response times in large - scale deployments [11].

3) Granular Control:
The library must provide granular control mechanisms, allowing developers to fine - tune feature activations for specific components within the project, ensuring a phased **and controlled rollout.**

4) Dependency Management:
Addressing dependencies between entities is crucial in large - scale projects. The library should facilitate the management of dependencies to prevent unforeseen issues during feature activation [12].

5) Configurability Across Levels:
Configuration options should be available at various levels, from project - wide settings to entity - specific configurations, providing flexibility in adapting to the diverse needs of a large - scale software system.

6) Monitoring and Analytics:
Implement robust monitoring and analytics capabilities to track the impact of feature rollouts, enabling real - time insights and the identification of potential issues in large - scale deployments[2].

7) Rollback Mechanism:
A reliable rollback mechanism should be in place, allowing for quick and efficient reverting of features in case unexpected issues arise during the rollout in large - scale environments [8].

Throughout this paper, we will address these criteria in the context of large - scale projects, illustrating how the proposed rollout configuration library meets these challenges head - on and empowers developers to navigate the complexities of feature deployments in expansive software ecosystems. We take an opiniated approach and explore large - scale feature rollout management through innovative and scalable solutions with code samples in C#[13].

2. System Overview

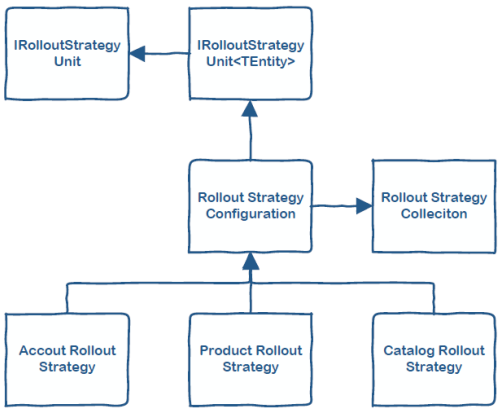

Figure 1: Rollout Strategy Configuration Overview

The rollout configuration library as illustrated in Figure 1, is designed to facilitate phased rollouts of features in software systems, offering a flexible and extensible framework. It enables developers to configure feature deployments based on the type of entity, providing fine - grained control over the introduction of new functionalities. The library is composed of several key components, offering a versatile solution for managing rollout strategies across entities[10].

***IRollout Strategy Unit* Interface**:
Represents a unit of rollout strategy associated with a specific entity. Defines properties like Key - entity identifier and Strategy (feature rollout strategy) [14].

```
publicinterfaceIRolloutStrategyUnit
{
 IEntity Key { get; }

IIdStrategy? Strategy { get; set; }
}
```

Also, provides a generic version for type - specific strategies *IRolloutStrategyUnit<TEntity>*[15]*:*

```
publicinterfaceIRolloutStrategyUnit<outTEntity
>: IRolloutStrategyUnit
whereTEntity: class, IEntity
{
newTEntity Key { get; }
}
```

***RolloutStrategyCollection* Class**:
Keyed collection that stores instances of IRolloutStrategyUnit. Facilitates efficient retrieval of rollout strategies based on entity keys. This class forms the backbone of the configuration system, providing a structured way to organize and manage rollout strategies.

```
public sealed
classRolloutStrategyCollection<TStrategyUnit>:
KeyedCollection<IEntity, TStrategyUnit>
whereTStrategyUnit: IRolloutStrategyUnit
{
publicRolloutStrategyCollection ()
{
}
protected override IEntity GetKeyForItem (TStrategyUnit
value)
 {
Ensure. ArgumentNotNull (value, nameof (value));
returnvalue. Key;
 }
}
```

***RolloutConfiguration* Class**:
Serves as a central configuration hub for rollout strategies. Utilizes a keyed collection (RolloutStrategyCollection) to organize and store rollout strategies for different entities. This allows developers to specify rollout configurations at various levels, such as project - wide settings and entity - specific configurations[1].

```
publicclassRolloutConfiguration<TStrategyUnit,
TStrategyCollection>
whereTStrategyUnit: IRolloutStrategyUnit
whereTStrategyCollection:
KeyedCollection<IEntity, TStrategyUnit>, new
()
{
publicTStrategyCollection
RolloutStrategyCollection
=>this. RolloutStrategyCollection;

public string Name =>this. Name;

}
```

***IRolloutStrategyCollectionProvider* Interface**:
Abstracts the retrieval of the rollout strategy collection for a given entity. This interface also enables a decoupled approach to providing rollout strategy collections, enhancing flexibility for different entities.

```
publicinterfaceIRolloutStrategyCollectionProvid
er
{
IFiRolloutStrategyCollectionGetRolloutStrategy
Collection (IEntity owner);
}
```

RolloutConfigurationUtils Class:
Contains utility methods for managing rollout strategies during migration. The methods in this classallow dynamic retrieval of the appropriate rollout strategy for a given entity, promoting generic and reusable migration logic[5].

```
publicstaticclassRolloutConfigurationUtils
{
publicstatic async Task<IIdStrategy>
GetMigrationStrategyAsync<TStrategyCollection
>
 (IModelReference modelReference,
RolloutStrategyCollectionProvider<TStrategyCol
lection>strategyCollectionProvider)
 where TStrategyCollection:
KeyedCollection<IEntity, IRolloutStrategyUnit>,
new ()
 {
if (! (modelReference. Entity is
IEntityWithOwnerentityWithOwner))
 {
returnnull;
 }

 IEntity ownerEntity = entityWithOwner.
OwnerEntity;

TStrategyCollectionstrategyCollection =
strategyCollectionProvider.
GetRolloutStrategyCollection (ownerEntity);

if (strategyCollection == null ||
strategyCollection. Count == 0)
 {
returnnull;
 }
```

```
 IRolloutStrategyUnit rolloutStrategy =
strategyCollection. TryGetValue
(modelReference. Entity, out var
entityRolloutStrategy)
 ? entityRolloutStrategy
: strategyCollection. TryGetValue (ownerEntity,
out IRolloutStrategyownerRolloutStrategy)
 ? ownerRolloutStrategy
: null;

return await Task. FromResult (rolloutStrategy?.
Strategy). ConfigureAwait (false);
 }
}
```

3. System Functionality

Configuration Flexibility:
The library empowers developers to configure feature rollouts at different levels, catering to both project - wide and entity - specific requirements. This flexibility enables the customization of rollout strategies based on the unique characteristics and dependencies of individual entities within the software ecosystem.

Entity - Centric Rollout:
The entity - centric approach ensures that rollout strategies can be associated with specific entities, tailoring feature deployments to the diverse needs of different components within a software system. This promotes a more granular and controlled rollout process[16].

Phased Rollout Management:
Developers can systematically manage the phased rollout of features using the *IRolloutStrategyUnit* interface. This enables a controlled activation of new functionalities, minimizing the risk of unforeseen issues during deployment [11].

How to Use:
Configuration Setup:
Instantiate a RolloutConfiguration object to serve as the central configuration hub. Use the *RolloutStrategyCollection* property to access the strategy collection and add rollout strategies for various types of entities or specific instances.

Entity - Specific Configuration:
Implement the IRolloutStrategyUnit interface for each entity requiring a custom rollout strategy. Specify the entity's key and the associated rollout strategy.

Phased Rollout in Migration:
Utilize the *GetMigrationStrategyAsync ()* method during migration to dynamically retrieve the appropriate rollout strategy for a given entity.

Implement the *IRolloutStrategyCollectionProvider* interface to provide rollout strategy collections tailored to different entities.

By following these steps, developers can leverage the generic and flexible nature of the rollout configuration library to achieve a tailored, entity - centric, and phased approach to feature deployments in their software systems.

4. Best Practices For Scalability

Scalability is a critical aspect of any software library, especially when dealing with large - scale systems. Below are some ideas and best practices to enhance the scalability of the rollout configuration library:

1) Distributed Configuration Store:
Store rollout configurations in a distributed and highly scalable data store, such as a distributed database or a cloud - based storage service. This allows for efficient retrieval of configuration data, reducing the load on a single point of access [4].

2) Caching Mechanism:
Implement a robust caching mechanism to store frequently accessed rollout configurations at the application level. This can significantly reduce the number of calls to the underlying data store, improving response times and minimizing latency during feature rollouts [17].

3) Configuration Sharding:
Shard the rollout configurations based on certain criteria, such as business unit, or entity type. This allows for parallelized access to configuration data, distributing the load across multiple shards and enhancing overall system scalability[4].

4) Horizontal Scaling of Configuration Services:
If the rollout configuration service is separate from the application, consider horizontally scaling it by deploying multiple instances. Load balancing mechanisms can distribute incoming requests among these instances, ensuring a balanced and scalable configuration service[9].

5) Optimized Rollout Strategy Collection Access:
Optimize the access patterns of the Rollout Strategy Collection by leveraging indexing or caching strategies. Efficient retrieval mechanisms can significantly reduce the time required to access rollout strategies, especially in scenarios involving large - scale collections [5] [11].

5. Conclusion

In conclusion, the rollout configuration library stands as a versatile and powerful tool, designed to help the way software teams approach feature deployments. Through the systematic management of phased rollouts based on entity types, the library empowers developers to achieve fine - grained control, flexibility, and scalability in the ever - evolving landscape of software development by taking an opiniated approach in implementing and configuring the library and showcase examples using C# [13][18] but is language agnostic.

The core components, such as:

- *IRollout Strategy Unit* interface,
- *Rollout Configuration*, and
- *Rollout Strategy Collection*

form a cohesive foundation that caters to the diverse needs of entities within a software system. The introduction of generics, coupled with the flexibility of the *IRollout Strategy Collection Provider* interface, ensures adaptability to different entities, configurations, and strategies.

In the pursuit of scalability, the library goes beyond conventional approaches, embracing groundbreaking ideas and best practices. From leveraging distributed configuration stores to implementing lazy loading and dynamic partitioning, the library is engineered to thrive in large - scale, distributed environments. These scalability enhancements, combined with a forward - thinking design, make the rollout configuration library a robust solution for the challenges posed by complex software ecosystems.

As development landscapes evolve and systems continue to grow in complexity, the rollout configuration library provides a roadmap for effective and scalable feature deployment. By following the outlined best practices and adopting the principles of flexibility, entity - centric rollout, and dynamic scalability, software teams can ensure a smoother, controlled, and adaptive transition to new functionalities.

In essence, the rollout configuration library is not just a tool; it's a strategic asset for software engineers aiming to navigate the intricate terrain of feature rollouts in modern, large - scale applications. Its impact goes beyond managing configurations; it sets the stage for a future where software deployment is not just a process but a well - orchestrated symphony of control, adaptability, and scalability.

References

[1] "Feature Toggles, " 2017. [Online]. Available: https: //martinfowler. com/articles/feature - toggles. html.
[2] T. Huff, "Developing, deploying, and supporting software according to the way of the cloud, " 2011. [Online]. Available: http: //highscalability. com/blog/2011/12/12/netflix - developing - deploying - and - supporting - software - accordi. html.
[3] K. H. Robert Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, Pearson.
[4] Kleppmann, Martin, Designing Data - Intensive Applications, O'Reilly Media, 2017.
[5] D. F. Jez Humble, in *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison - Wesley Signature Series, 2010.
[6] O. Dijkstra, *Extending the agile development discipline to deployment: The need for a holistic approach*, Utrecht University, 2013.
[7] K. Beck, Extreme Programming Explained: Embrace Change, Addison - Wesley, 2000.
[8] M. Brittain, "Continuous deployment: The dirty details, " [Online]. Available: http: //www.slideshare. net/mikebrittain/mbrittain - continuous - deploymentalm3public.
[9] J. A. a. P. Hammond, "10 deploys per day - Dev and Ops cooperation at Flickr, " [Online]. Available: ttp: //www.slideshare. net/jallspaw/10 - deploys - per - day - dev - and - ops - cooperation - at - flickr.
[10] M. D. M. G. L. W. K. B. M. S. Tony Savor, "Continuous deployment at Facebook and OANDA, " *ICSE '16: Proceedings of the 38th International Conference on Software Engineering Companion*, pp.21 - 30, 2016.
[11] B. Schmaus, "Deploying the Netflix API, " 2013. [Online]. Available: http: //techblog. netflix. com/2013/08/deploying - netflix - api. html.
[12] M. Mullapudi, "OBJECT ORIENTED CONCEPTS, " 2016. [Online]. Available: https: //tutorialq. com/java/object - oriented - concepts/.
[13] "Microsoft csharp, " [Online]. Available: https: //learn. microsoft. com/en - us/dotnet/csharp/language - reference/.
[14] M. Mullapudi, "ACCESS AND NON - ACCESS MODIFIERS, " 2016. [Online]. Available: https: //tutorialq. com/java/object - oriented - concepts/access - and - non - access - modifiers/.
[15] "Generic classes and methods, " [Online]. Available: https: //learn. microsoft. com/en - us/dotnet/csharp/fundamentals/types/generics.
[16] H. E. A. C. B. H. G. M. G. A. H. J. M. J. M. B. S. T. S. M. W. S. a. W. L. Parnin C, "The Top 10 Adages in Continuous Deployment, " *IEEE Software*, vol.34, no.3, pp.86 - 95, 2017.
[17] "Redis cache, " [Online]. Available: https: //redis. io/.
[18] M. Mullapudi, "HOW TO LEARN CSHARP, " 2016. [Online]. Available: https: //tutorialq. com/programming/csharp.