redhat.

Drools: Rule engines in the microservices era

Mario Fusco – Drools Project Lead
Donato Marrazzo – Senior Solutions Architect
Matteo Mortari – Senior Software Engineer

#RedHatOSD

Business Rules are at the heart of every organization

Laws, Regulations and Policies

Drive Process Decisions

Determine Pricing of Products and Services

Business Rules – An Overview

Separate decision logic from application code

Write once, use anywhere. Agile rule lifecycle management.

Decision logic defined in business terminology and language

Domain experts directly involved in rule definition and writing.

Performance and scalability

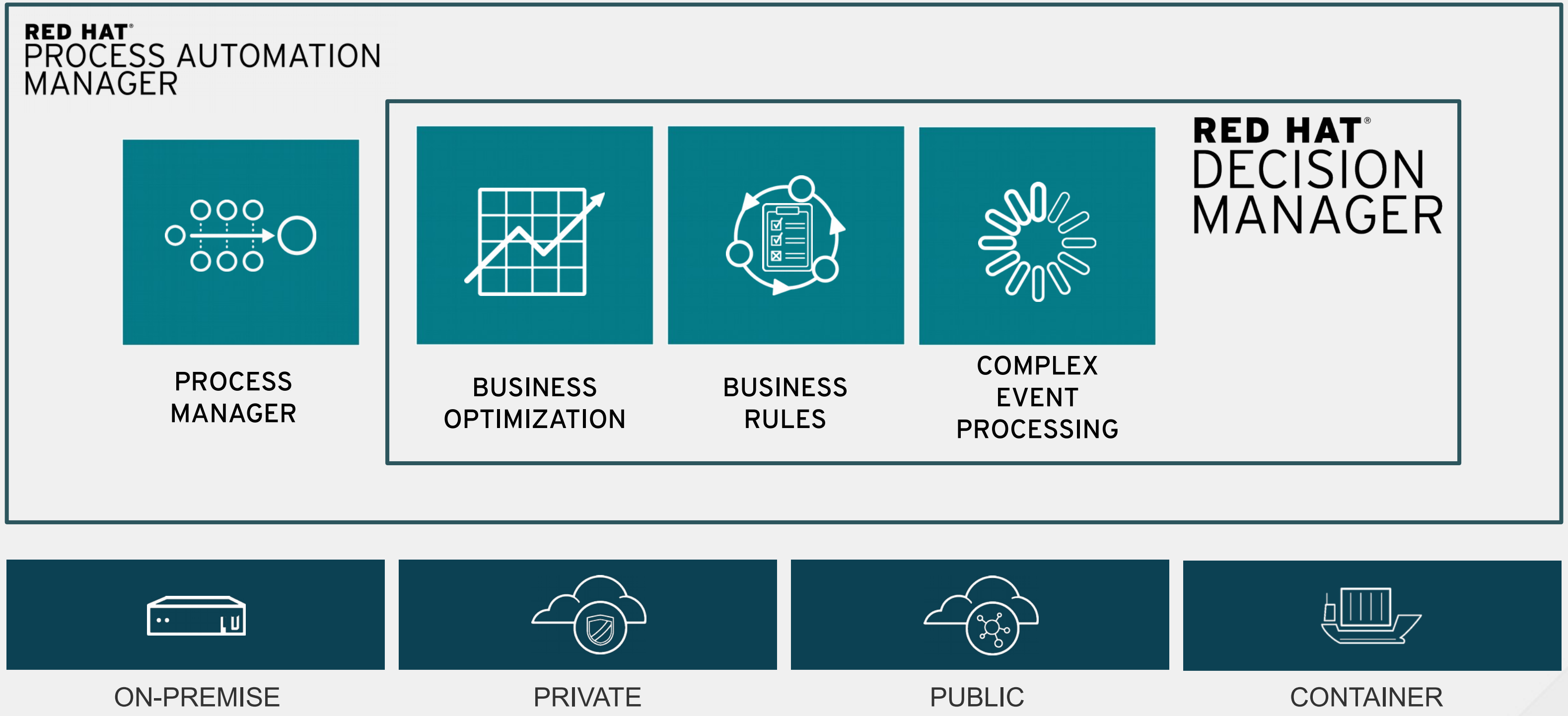
From 10 to 1,000,000 rules.

Decision Management and Automation Value across industries

Banking	Insurance	Capital Markets	Public Sector
Loan Organisation	Claims Processing	Automated Trading	Claims Processing
Credit Decisioning	Underwriting	Trade Order Mgmt	Entitlement Calc.
Sales Advisory	Quoting	Accounting	Benefit Calc.
Payments	Rating	Compliance	Fraud Detection
Accounting	Commissioning	KYC/AML	Screening
		On Boarding	

Telecom	Transportation	Retail	Manufacturing
Offer Configuration	Promotions Mgmt	Recommendation	Order Mgmt
Order Mgmt	Loyalty Programs	Campaign Mgmt	Billing
Fraud Detection	Customer Service	Order Mgmt	Contract Mgmt
Loyalty Programs	Billing	Pricing	
Network Monitoring	Contract Mgmt		

Red Hat automation products



New in Drools 7 Executable Model

- A pure Java DSL for Drools rules authoring
- Automatically generated by maven plugin
- Can be embedded in kjar
  - Faster compilation
  - Backward/Forward compatible
- Allow for faster prototyping and experiment of new features

```
Result result = new Result();
Variable<Person> markV = declarationOf( Person.class );
Variable<Person> olderV = declarationOf( Person.class );

Rule rule = rule( "beta" )
      .build(
            pattern(markV)
                        .expr("exprA", p -> p.getName().equals( "Mark" ),
                              alphaIndexedBy( String.class, ConstraintType.EQUAL, 1, p -> p.getName(), "Mark" ),
                              reactOn( "name", "age" )),
            pattern(olderV)
                        .expr("exprB", p -> !p.getName().equals("Mark"),
                              alphaIndexedBy( String.class, ConstraintType.NOT_EQUAL, 1, p -> p.getName(), "Mark" ),
                              reactOn( "name" ))
                        .expr("exprC", markV, (p1, p2) -> p1.getAge() > p2.getAge(),
                              betaIndexedBy( int.class, ConstraintType.GREATER_THAN, 0, p -> p.getAge(), p ->
p.getAge() ),
                              reactOn( "age" )),
            on(olderV, markV).execute((p1, p2) -> result.setValue( p1.getName() + " is older than " + p2.getName()))
      );
```

New in Drools 7
Rule Units

- Declarative approach to:
  - Partition a rules set into smaller units.
  - Binding datasources to a unit.
  - Orchestrate the execution of a unit.
- Aggregate of a data-source, global variables and rules.
- Better coupling between data and rules acting on that specific data

```
package org.mypackage.myunit
unit AdultUnit

rule Adult when
    $p : Person(age >= adultAge) from persons
then
    System.out.println($p.getName() + " is adult and greater than " + adultAge);
end
```

```
package org.mypackage.myunit;

public static class AdultUnit implements RuleUnit {
    private int adultAge;
    private DataSource<Person> persons;

    public AdultUnit( ) { }

    public AdultUnit( DataSource<Person> persons, int age ) {
        this.persons = persons;
        this.age = age;
    }

    // A DataSource of Persons for this RuleUnit
    public DataSource<Person> getPersons() {
        return persons;
    }

    // A global variable valid in this RuleUnit
    public int getAdultAge() {
        return adultAge;
    }

    // --- life cycle methods

    @Override
    public void onStart() {
        System.out.println(getName() + " started.");
    }

    @Override
    public void onEnd() {
        System.out.println(getName() + " ended.");
    }
}
```

Business Optimizer

Optimize **Goals**

With limited **Resources**

Under **Constraints**

Need for Standards in Decision Management space

- Decisions are a common language across business, IT and analytic organizations improving collaboration, increasing reuse, and easing implementation.
- Business analysts wish to model and improve the decisions that their businesses make.
- Common notation which is understandable by all business users.
- Standardized bridge between the decision design and implementation.
- Usable alongside BPMN business process notation.
- Rules are just a portion of the logic needed to make a decision.

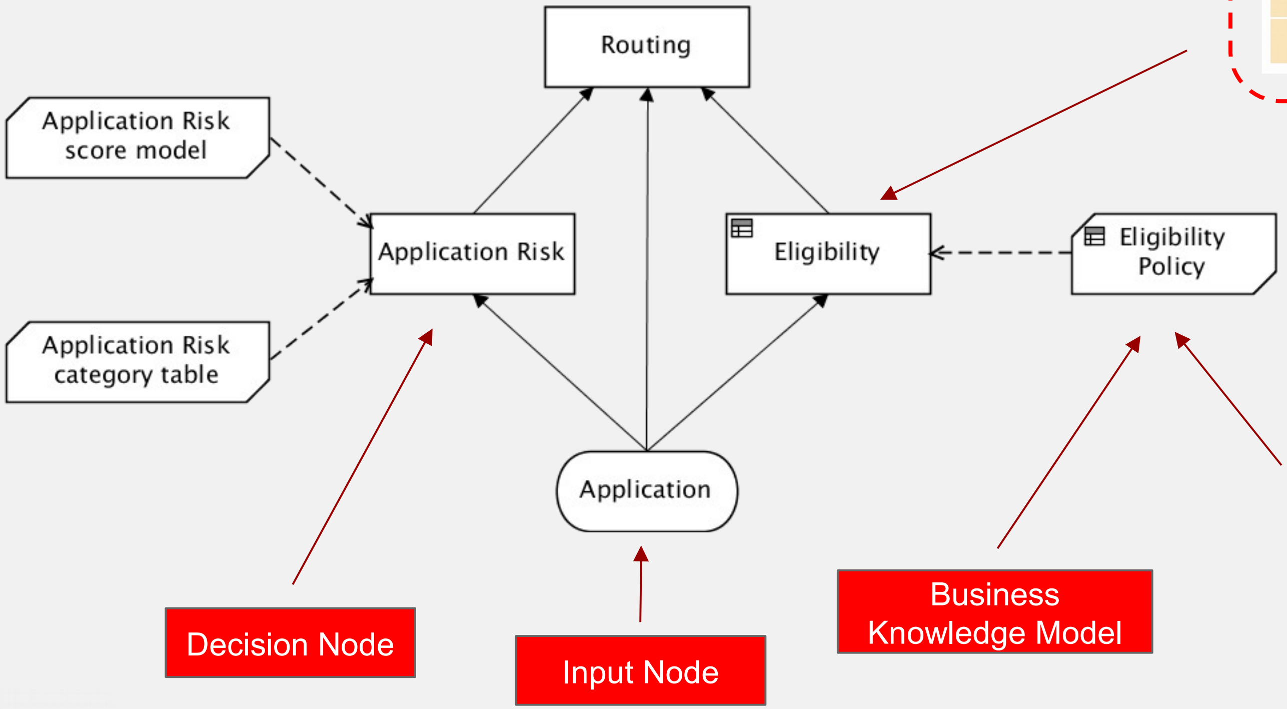
What is DMN?

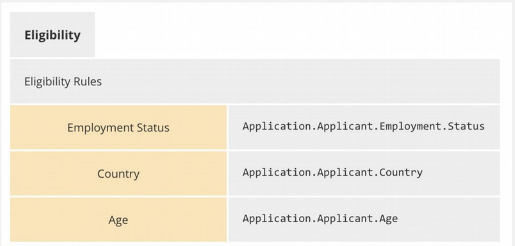
DMN, which stands for Decision Model and Notation, is a relatively new standard managed by OMG, the organization behind BPMN. It is trying to do for Business Decision Management what BPMN did for Business Process Management a decade ago: empower the business to take charge of the logic that drives its operations, through a vendor-independent diagramming language.

– Bruce Silver, http://methodandstyle.com/what-is-dmn

Decision Model and Notation

Example:

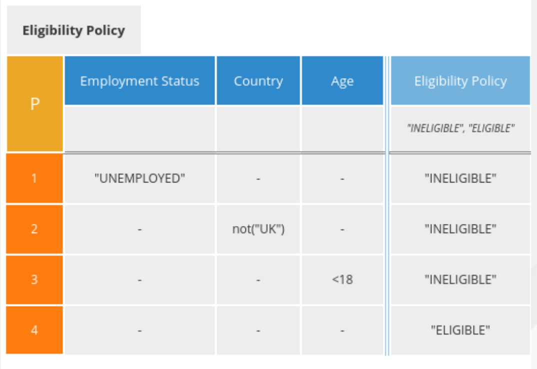




Eligibility

Eligibility Rules

Employment Status	`Application.Applicant.Employment.Status`
Country	`Application.Applicant.Country`
Age	`Application.Applicant.Age`

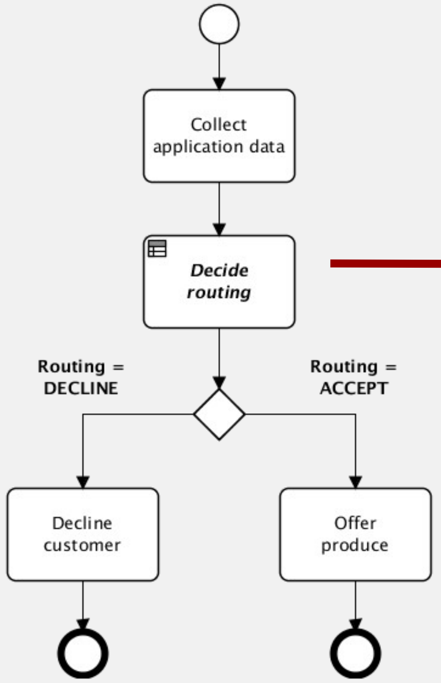


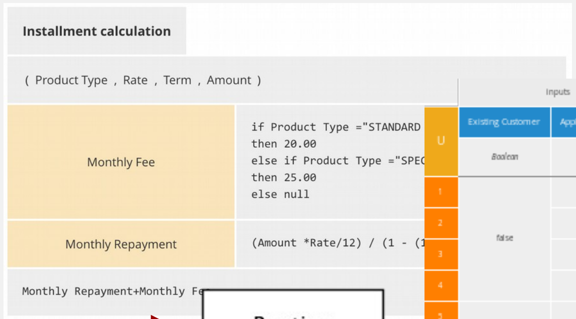
Eligibility Policy

P	Employment Status	Country	Age	Eligibility Policy
				"INELIGIBLE", "ELIGIBLE"
1	"UNEMPLOYED"	-	-	"INELIGIBLE"
2	-	not("UK")	-	"INELIGIBLE"
3	-	-	<18	"INELIGIBLE"
4	-	-	-	"ELIGIBLE"

DMN Big Picture

DMN in context of BPMN



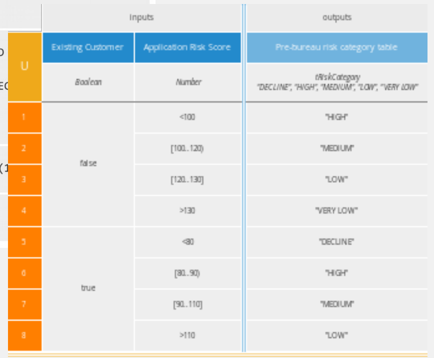


Installment calculation

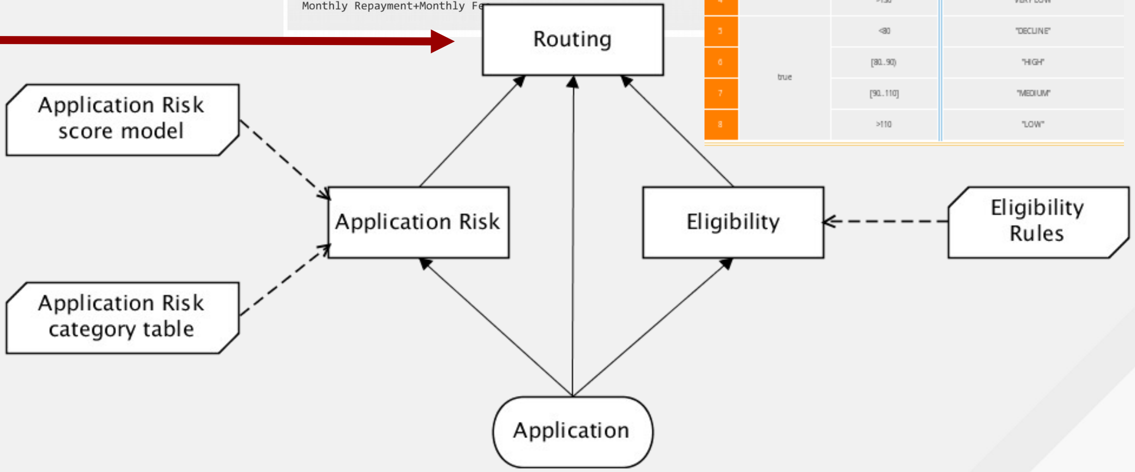
(Product Type , Rate , Term , Amount)

Monthly Fee	`if Product Type ="STANDARD` `then 20.00` `else if Product Type ="SPEC` `then 25.00` `else null`
Monthly Repayment	`(Amount *Rate/12) / (1 - (1`

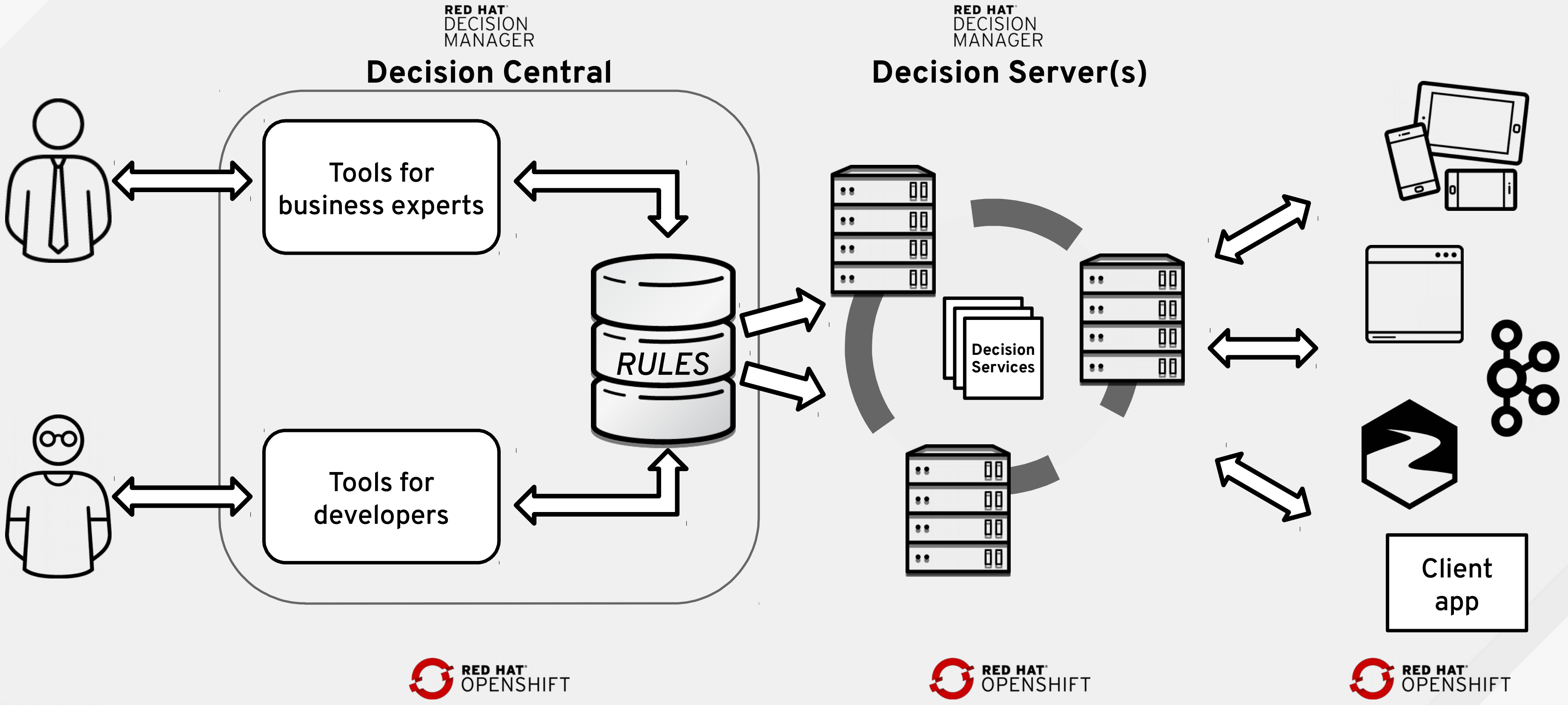
```
Monthly Repayment+Monthly Fe
```



	Inputs		outputs
U	Existing Customer	Application Risk Score	Pre-bureau risk category table
	Boolean	Number	tRiskCategory "DECLINE", "HIGH", "MEDIUM", "LOW", "VERY LOW"
1	false	<100	"HIGH"
2		[100..120)	"MEDIUM"
3		[120..130]	"LOW"
4		>130	"VERY LOW"
5	true	<80	"DECLINE"
6		[80..90)	"HIGH"
7		[90..110]	"MEDIUM"
8		>110	"LOW"



Architecture example



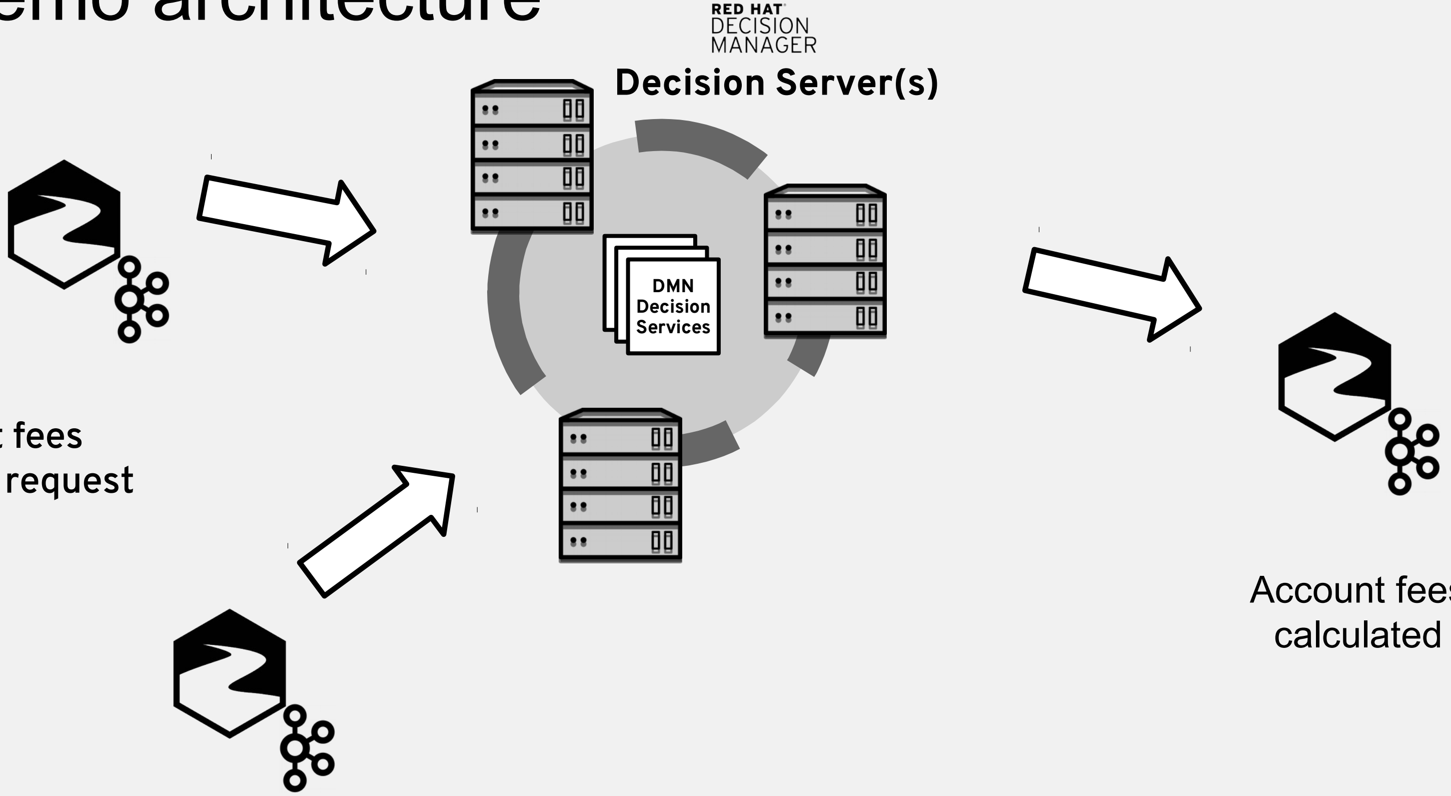
Author / Test

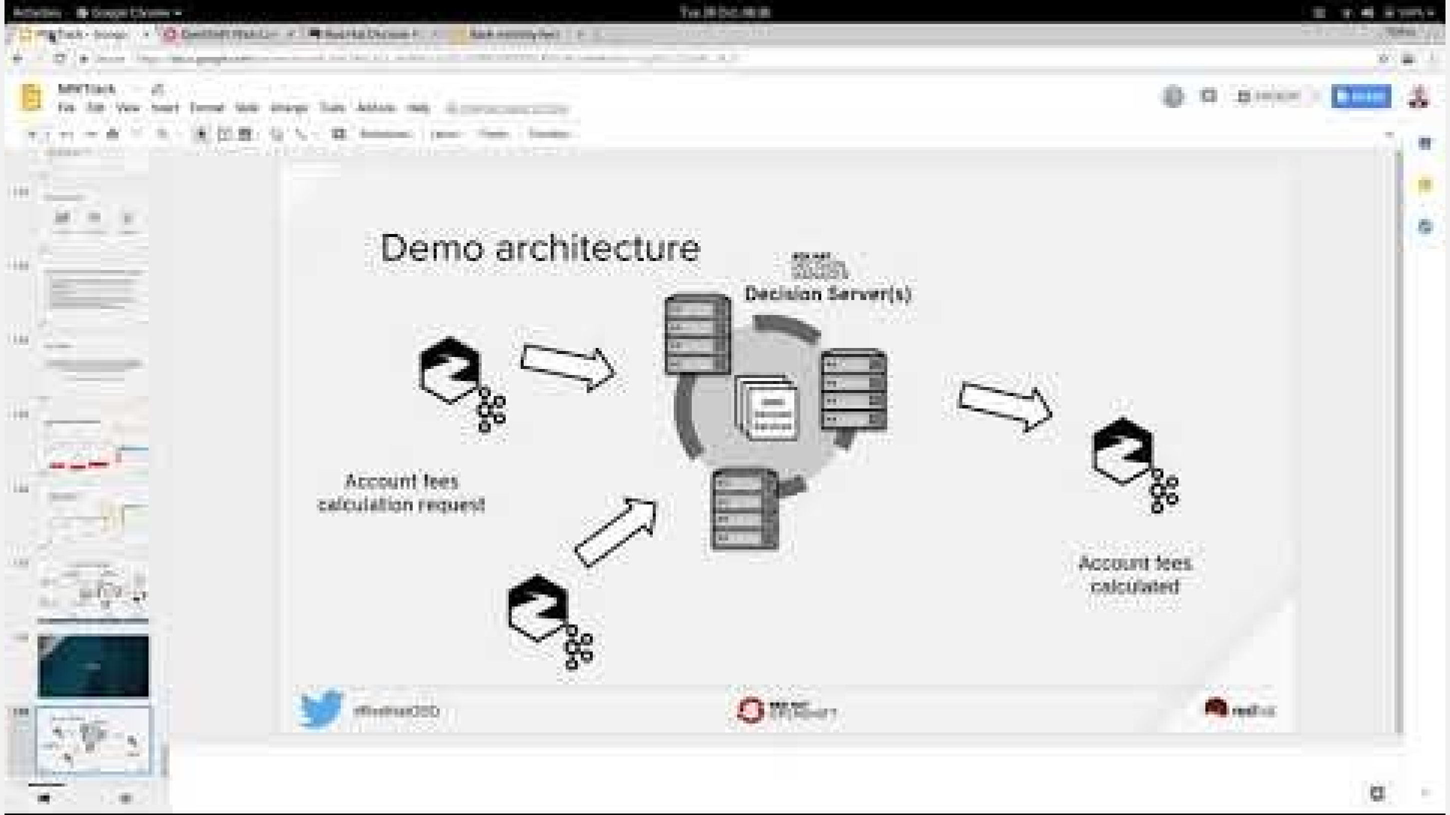
Deploy

Expose

Demo

Demo architecture





Demo architecture

redhat.

GRAZIE PER L'ATTENZIONE !

Mario Fusco – Drools Project Lead
Donato Marrazzo – Senior Solutions Architect
Matteo Mortari – Senior Software Engineer

#RedHatOSD