Muhammad Uzair

- ➢ **PAINLESS CONTEXT:**

ainless scripts run inside an Elasticsearch operation like an update or search. They can access different kinds of data depending on the context, such as the original document (`_source`), indexed fields (`doc`), or parameters you've defined (`params`). Scripts can only use the data provided by their specific context and can't directly access internal Elasticsearch data.

the **Painless Field** context provides a script with access to three types of data:

- **`doc`**: This variable allows you to access data from fields that are part of Elasticsearch's internal data structures. For example, `doc['credit_score']` would get the value of the `credit_score` field.
- **`params['_source']`**: This variable gives you access to the original, raw source of the document. For instance, `params['_source']['name']` would retrieve the name directly from the document's `_source` field.
- **`params`**: This is a map that holds any user-defined parameters you've passed in with your query, such as `params['good_interest_rate']`

In runtime field type (composite) actually tells the query that its returning multiple fields

LOAN INTEREST RATE APP

```
GET borrowers/_search
{
  "fields": [
     "loan_quote.loan_id", "loan_quote.interest_rate", "loan_quote.risk_rating"
],
  "runtime_mappings": {
     "loan_quote": {
       "type": "composite",
       "script": {
         "lang": "painless",
         "source": """
           String loanId = params['_source']['name'] + "-" + params['_source']['id'];
           if (doc['credit_rating'].value > params.risk_threshold) {
             emit(["loan_id": loanId, "risk_rating": "Good", "interest_rate": params['good_interest_rate']]);
           } else {
             emit(["loan_id": loanId, "risk_rating": "Poor", "interest_rate": params['poor_interest_rate']]);
           }
         """,
         "params": {
            "risk_threshold": 500, "good_interest_rate": 0.10, "poor_interest_rate": 0.15
         }
     },
     "fields": {
        "loan_id": {
          "type": "keyword"
```

WE WILL USE RUNTIME FIELDS CONCEPT THERE

TEST CODE:

```
PUT borrowers/_doc/1
{
  "id": "001",
  "name": "Joe",
  "credit_rating": 450
}

PUT borrowers/_doc/2
{
  "id": "002",
  "name": "marry",
  "credit_rating": 650
}

GET borrowers/_search
{
  "_source": false,
  "fields": [
    "name",
    "id",
    "loan_quote.loan_id",
    "loan_quote.interest_rate",
    "loan_quote.risk_rating"
    ],
    "runtime_mappings": {
      "loan_quote": {
        "type": "composite",
        "script": {
          "lang": "painless",
          "source": "emit(['loan_id': 'bob-3' , 'risk_rating': 'Good' , 'interest_rate': 0.1])"
        },
        "fields": {
          "loan_id": {
            "type": "keyword"
            },
          "interest_rate": {
            "type": "double"
          },
          "risk_rating": {
            "type": "keyword"
          }
        }
```

```
    }
  }
}
```

```
PUT borrowers/_doc/1
{
  "id": "001",
  "name": "Joe",
  "credit_rating": 450
}

PUT borrowers/_doc/2
{
  "id": "002",
  "name": "marry",
  "credit_rating": 650
}
```

```
GET borrowers/_search
{
  "_source": false,
  "fields": [
    "name",
    "id",
    "loan_quote.loan_id",
    "loan_quote.interest_rate",
    "loan_quote.risk_rating"
    ],
    "runtime_mappings": {
      "loan_quote": {
        "type": "composite",
        "script": {
          "lang": "painless",
          "source": "emit(['loan_id': 'bob-3' , 'risk_rating': 'Good' , 'interest_rate': 0.1])"
        },
        "fields": {
          "loan_id": {
            "type": "keyword"
            },
          "interest_rate": {
            "type": "double"
          },
          "risk_rating": {
            "type": "keyword"
          }
        }
      }
    }
}
```

OUTPUT:

```
{
  "took": 24,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 2,
      "relation": "eq"
    },
    "max_score": 1,
    "hits": [
      {
        "_index": "borrowers",
        "_id": "1",
        "_score": 1,
        "fields": {
          "name": [
            "Joe"
          ],
          "loan_quote.risk_rating": [
            "Good"
          ],
          "loan_quote.interest_rate": [
            0.1
          ],
          "id": [
            "001"
```

```
        ],
        "loan_quote.loan_id": [
          "bob-3"
        ]
      }
    },
    {
      "_index": "borrowers",
      "_id": "2",
      "_score": 1,
      "fields": {
        "name": [
          "marr"
        ],
        "loan_quote.risk_rating": [
          "Good"
        ],
        "loan_quote.interest_rate": [
          0.1
        ],
        "id": [
          "002"
        ],
        "loan_quote.loan_id": [
          "bob-3"
        ]
      }
    }
  ]
 }
}
```


NOW THE TESTING IS COMPLETE NOW WE WILL WRITE ACTUAL CODE AS WE HAVE SUCCESSFULLY CREATED THE STRUCUTRE THAT IS SHOWING CORRECT OUTPUT

CASE 1(Inline runtime field script.)

ACTUAL CODE:

```
PUT borrowers/_doc/1
{
  "id": "001",
  "name": "Joe",
  "credit_rating": 450
}

PUT borrowers/_doc/2
{
  "id": "002",
  "name": "marry",
  "credit_rating": 650
}
```

```
GET borrowers/_search
{
  "_source": false,
  "fields": [
    "name",
    "id",
    "loan_quote.loan_id",
    "loan_quote.interest_rate",
    "loan_quote.risk_rating"
    ],
    "runtime_mappings": {
      "loan_quote": {
        "type": "composite",
        "script": {
          "lang": "painless",
          "source": """
            String loanId = doc['name.keyword'].value + "-" + doc["id.keyword"].value;
            if (doc['credit_rating'].value > params.threshold) {
              emit(["loan_id": loanId, 'risk_rating': 'Good', 'interest_rate': params.good_interest_rate]);
            } else {
              emit(["loan_id": loanId, 'risk_rating': 'Poor', 'interest_rate': params.poor_interest_rate]);
            }
          """,
          "params": {
            "threshold": 500,
            "good_interest_rate": 0.1,
            "poor_interest_rate": 0.15
          }
        },
        "fields": {
          "loan_id": {
            "type": "keyword"
            },
          "interest_rate": {
            "type": "double"
          },
          "risk_rating": {
            "type": "keyword"
          }
        }
      }
    }
}
```

```
PUT borrowers/_doc/1
{
  "id": "001",
  "name": "Joe",
  "credit_rating": 450
}

PUT borrowers/_doc/2
{
  "id": "002",
  "name": "marry",
  "credit_rating": 650
}
```

```
GET borrowers/_search
{
  "_source": false,
  "fields": [
    "name",
    "id",
    "loan_quote.loan_id",
    "loan_quote.interest_rate",
    "loan_quote.risk_rating"
    ],
    "runtime_mappings": {
      "loan_quote": {
        "type": "composite",
        "script": {
          "lang": "painless",
          "source": """
            String loanId = doc['name.keyword'].value + "-" + doc["id.keyword"].value;
            if (doc['credit_rating'].value > params.threshold) {
              emit(["loan_id": loanId, 'risk_rating': 'Good', 'interest_rate': params.good_interest_rate]);
            } else {
              emit(["loan_id": loanId, 'risk_rating': 'Poor', 'interest_rate': params.poor_interest_rate]);
            }
          """,
          "params": {
            "threshold": 500,
            "good_interest_rate": 0.1,
            "poor_interest_rate": 0.15
          }
        },
        "fields": {
          "loan_id": {
            "type": "keyword"
            },
          "interest_rate": {
            "type": "double"
          },
          "risk_rating": {
            "type": "keyword"
          }
        }
      }
    }
}
```

OUTPUT:

```
{
  "took": 1,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 2,
      "relation": "eq"
    },
    "max_score": 1,
    "hits": [
      {
        "_index": "borrowers",
        "_id": "1",
        "_score": 1,
        "fields": {
          "loan_quote.interest_rate": [
            0.15
          ],
          "loan_quote.loan_id": [
            "Joe-001"
          ],
          "name": [
            "Joe"
          ],
          "loan_quote.risk_rating": [
            "Poor"
          ],
          "id": [
            "001"
          ]
        }
      },
      {
        "_index": "borrowers",
        "_id": "2",
        "_score": 1,
        "fields": {
          "loan_quote.interest_rate": [
            0.1
          ],
          "loan_quote.loan_id": [
            "marr-002"
          ],
          "name": [
            "marr"
          ],
          "loan_quote.risk_rating": [
            "Good"
          ],
          "id": [
            "002"
          ]
        }
      }
    ]
  }
}
```

Characteristics:

- Logic is written **inline** inside the script body.
- Shorter, simpler, good for small scripts.
- No custom function is defined — all logic executes directly in the script.
- Easier to read for **quick one-off runtime mappings**.

CASE 2(Function-based runtime field script.)

CODE:

```
PUT borrowers/_doc/1
{
  "id": "001",
  "name": "Joe",
  "credit_rating": 450
}

PUT borrowers/_doc/2
{
  "id": "002",
  "name": "marry",
  "credit_rating": 650
}

GET borrowers/_search
{
  "_source": false,
  "fields": [
    "name",
    "id",
    "loan_quote.loan_id",
    "loan_quote.interest_rate",
    "loan_quote.risk_rating"
    ],
    "runtime_mappings": {
      "loan_quote": {
        "type": "composite",
```

```
      "script": {
        "lang": "painless",
        "source": """
        Map getLoanQuote (String id, String name, long creditRating, double riskThreshold,
double goodInterestRate,    double poorInterestRate) {
          String loanId = name + "-" + id;
          String riskAssessment = "";
          double interestRate = 0.0;
          if (creditRating > riskThreshold) {
           riskAssessment = "Good";
           interestRate = goodInterestRate;
          } else {
            riskAssessment = "Poor";
            interestRate = poorInterestRate;
          }
          return ["loan_id": loanId, "risk_assessment": riskAssessment, "interest_rate":
interestRate];
        }
        emit(
        getLoanQuote(doc['id.keyword'].value , doc['name.keyword'].value ,
doc['credit_rating'].value , params.threshold , params.good_interest_rate ,
params.poor_interest_rate)
        );
       """,
       "params": {
        "threshold": 500,
        "good_interest_rate": 0.1,
        "poor_interest_rate": 0.15
       }
      },
      "fields": {
       "loan_id": {
        "type": "keyword"
        },
       "interest_rate": {
        "type": "double"
       },
       "risk_rating": {
        "type": "keyword"
       }
      }
    }
  }
}
```

```
PUT borrowers/_doc/1
{
  "id": "001",
  "name": "Joe",
  "credit_rating": 450
}

PUT borrowers/_doc/2
{
  "id": "002",
  "name": "marry",
  "credit_rating": 650
}
```

```
GET borrowers/_search
{
  "_source": false,
  "fields": [
    "name",
    "id",
    "loan_quote.loan_id",
    "loan_quote.interest_rate",
    "loan_quote.risk_rating"
    ],
    "runtime_mappings": {
      "loan_quote": {
        "type": "composite",
        "script": {
          "lang": "painless",
          "source": """
          Map getLoanQuote (String id, String name, long creditRating, double riskThreshold, double goodInterestRate,
             double poorInterestRate) {
            String loanId = name + "-" + id;
            String riskAssessment = "";
            double interestRate = 0.0;
            if (creditRating > riskThreshold) {
              riskAssessment = "Good";
              interestRate = goodInterestRate;
            } else {
                riskAssessment = "Poor";
                interestRate = poorInterestRate;
            }
            return ["loan_id": loanId, "risk_assessment": riskAssessment, "interest_rate": interestRate];
          }
          emit(
            getLoanQuote(doc['id.keyword'].value , doc['name.keyword'].value , doc['credit_rating'].value , params
              .threshold , params.good_interest_rate , params.poor_interest_rate)
            );
          """,
          "params": {
            "threshold": 500,
            "good_interest_rate": 0.1,
            "poor_interest_rate": 0.15
          }
        },
        "fields": {
          "loan_id": {
            "type": "keyword"
            },
          "interest_rate": {
            "type": "double"
          },
          "risk_rating": {
            "type": "keyword"
          }
        }
      }
    }
}
```

OUTPUT:

```
{
  "took": 35,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 2,
      "relation": "eq"
    },
    "max_score": 1,
    "hits": [
      {
        "_index": "borrowers",
        "_id": "1",
        "_score": 1,
        "fields": {
          "loan_quote.interest_rate": [
            0.15
          ],
          "loan_quote.loan_id": [
            "Joe-001"
          ],
          "name": [
            "Joe"
          ],
          "id": [
            "001"
          ]
        }
      },
      {
        "_index": "borrowers",
        "_id": "2",
        "_score": 1,
        "fields": {
          "loan_quote.interest_rate": [
            0.1
          ],
          "loan_quote.loan_id": [
            "marr-002"
          ],
          "name": [
            "marr"
          ],
          "id": [
            "002"
          ]
        }
      }
    ]
  }
}
```

Characteristics:

- Defines a **reusable function** `getLoanQuote`.
- Logic is **modularized** → easier to maintain if the calculation grows complex.
- Allows **clean separation** of data extraction vs calculation.
- More verbose than Case 1, but scalable.

➢ **UPDATING A DOCUMENT:**

If we a are updating a document in elastic search and we want to do script in there

What we are doing is taking a fresh copy of the source and we are about to reindex it, well there is a context in which we can sue the script to modify the source before it gets indexed. This is the only time when we can actually modify the source

UPDATING THE CREDIT RATING:

Lets say that the ank wants to update borrowers and they wan tto adjust the credit raring by certain percentage so in this case we can write a script that woul take the update percent from the bank and figure out a formula so we can update it, so we can up or it or lower it by that amount. In this case we dont want to do a function, it make sense to save the script because even though we are going to even though the script can only be saved and reused if its using the context data. But in this case we can just assume that the script will be used only in the carious update context meaning the data will come from the source. So if that is the case then we can create the script using the script API and then call it by ID

FOLLOWING SCRIPT UPDATE THE CREDIT RATING OF SINGLE BORROWER WHEN USED WITHIN AN UPDATE CONTEXT

```
PUT _scripts/update_credit_rating
{
  "script": {
    "lang": "painless",
    "source": """
      double updateFactor = 1.0 + params['update_percent'] / 100.0;
      int creditRating = ctx['_source']['credit_rating'];
      ctx['_source']['credit_rating'] = (int)(creditRating * updateFactor);
    """
  }
}
```

WE CAN ALSO UPDATE ALL THE BORROWERS IF WE INVOKE THE SAME SCRIPT WITHIN AN _update_by_query

```
POST borrowers/_update_by_query
{
  "script": {
    "id": "update_credit_rating",
    "params": {
      "update_percent": 10
    }
  }
}
```

WE CAN USE THE SCRIPT TO CREATE new_borrowe AFTER UPDATING EVERY borrower's credit rating using _reindex

```
POST _reindex
{
   "source": {
     "index": "borrowers"
   },
   "dest": {
     "index": "new_borrowers"
   },
   "script": {
     "id": "update_credit_rating",
     "params": {
       "update_percent": 10
     }
   }
}
```

- **CUSTOM FILTERS:**

Lets say we are in a query and the match logic that elasticsearch gives us is not enough for the filters we have in mind So we want customs filters with a boolean expression that says if the document have these fields and they are related in this way then its a hit if they are not then its not a hit

EXAMPLE:

Lets say bank wants to find borrowers with credit rating greater than Bank;s average rating multiplied by some factor. So they will provide the average rating and the multiplier. In this case we need to look up filter context

THE QUERY CAN BE WRITTEN AS FOLLOW USING THE SCRIPTED FILTER

```
GET borrowers/_search
{
  "query": {
    "bool": {
      "filter": {
        "script": {
          "script": {
            "lang": "painless",
              "source": """
                long creditRating = doc['credit_rating'].value;
                return (creditRating > (params['multiplier'] * params['bank_avg_rating']));
              """,
              "params": {
                "bank_avg_rating": 525,
                "multiplier": 1.23
               }
             }
           }
         }
      }
    }
}
```

- **CUSTOM SORT:**

What if we want to change the order in which documents are returned. Like We are doing a full text search documents are sorted in sending order by by relevant score. But we want some other field in the document to affect the score. So we can do tht with scripting

Example:

The bank added a field to the borrowers index containing there current loan amount so it keeps track of how much money that have loan from the bank. So we wan to rank them in descending order by there risk rating.

WE will simply use following _update statement to update our data

```
POST borrowers/_update/1
{
  "doc": {
    "current_loan_total": 100000
  }
}

POST borrowers/_update/2
{
  "doc": {
    "current_loan_total": 5000
  }
}
```

EXAMPLE 2:

Bank would like a query ranking borrowers using the formula

- Credit_rating / 1000.0 * current_loan_total / 1000 * adjustment

We can sort the result of a query using this formula by embedding the script in the sort clause of _search

We will use the painless Sort context which exposes document data through the **doc** local variable, plus any user-defined parameter

A QUERY THAT ORDER BORROWERS AS REQUESTED BY THE BANK CAN BE WRITTEN USING SCRIPTED SORT CLAUSE AS FOLLOW

```
GET borrowers/_search
{
  "query": {
    "match_all": {}
},
"sort": [
  {
   "_script": {
     "type": "number",
     "script": {
       "lang": "painless",
       "source": """
         double creditFactor = doc['credit_rating'].value / 1000.0;
         double currentLoanFactor = doc['current_loan_total' ].value / 1000.0;
         double adjustment = params['adjustment'];
         return creditFactor * currentLoanFactor * adjustment; "
         "",
         "params": {
           "adjustment": 2.1
         }
     },
     "order": "desc"
    }
   }
  ]
}
```

THE VALUE OF THE FORMULA CALCULATED BY OUR SCRIPT IS RETURNED IN THE RESULT, SO WE GET BORROWERS RANKING AS WELL, NOTE HOW BORROWER JOE IS RANKED HIGHER THAN THAN MERRY BECAUSE OF THERE CURRENT LOANS

```
"hits" : [
  {
    "_index" : "borrowers",
    "_id" : "1",
    "_score" : null,
    "_source" : {
      "id" : "001",
      "name" : "Joe",
      "credit_rating" : 450,
      "current_loan_total" : 100000
    },
    "sort" : [
      94.5
    ]
  },
  {
    "_index" : "borrowers",
    "_id" : "2",
    "_score" : null,
    "_source" : {
    "id" : "002",
    "name" : "Mary",
    "credit_rating" : 650,
    "current_loan_total" : 5000
  },
  "sort" : [
```


Key Takeaways

- Painless scripts access data via **doc, params, and _source**.
- Runtime fields allow dynamic computed fields (inline or function-based).
- Scripts can update documents, define custom filters, and apply custom sorting.
- Inline scripts are simpler; function-based scripts are more maintainable for complex logic.