Pulsar Functions

A Deep Dive | Pulsar Summit 2020

Sanjeev Kulkarni
sanjeevk@splunk.com

splunk> turn data into doing™

Pulsar Functions:- A Deep Dive

Agenda

Brief introduction to Pulsar Functions

Deep Dive into internals

- Submission workflow
- Scheduling workflow
- Execution workflow
- Java Instance concepts

Current/Future Work

Pulsar Functions:- A Deep Dive

Agenda

Brief introduction to Pulsar Functions

Deep Dive into internals

- Submission workflow
- Scheduling workflow
- Execution workflow
- Java Instance concepts

Current/Future Work

Pulsar Functions:- A Brief Introduction

Core Concept

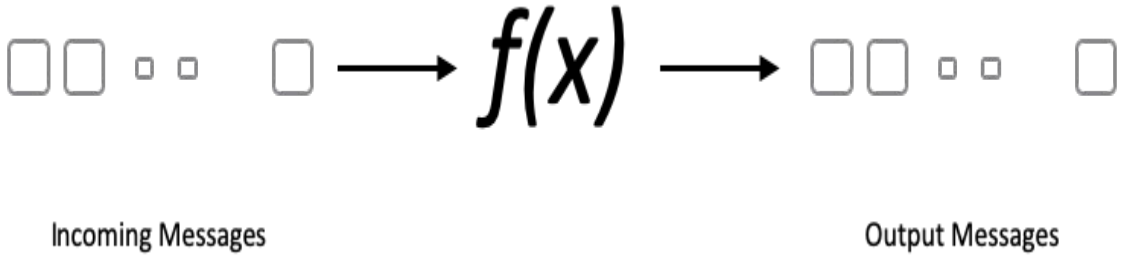
Bringing Serverless concepts to the streaming world.

Execute processing logic per message on input topic

Function output goes to an output topic

- Optional

Abstract View



Pulsar Functions:- A Brief Introduction

Simple API

Emphasis on simplicity

Great for 90% use-cases on streams

- Filtering
- Routing
- Enrichment

Not meant to replace Spark/Flink

SDK-less API

```
import java.util.function.Function;
public class ExclamationFunction implements Function<String, String> {
    @Override
    public String apply(String input) {
        return input + "!";
    }
}
```

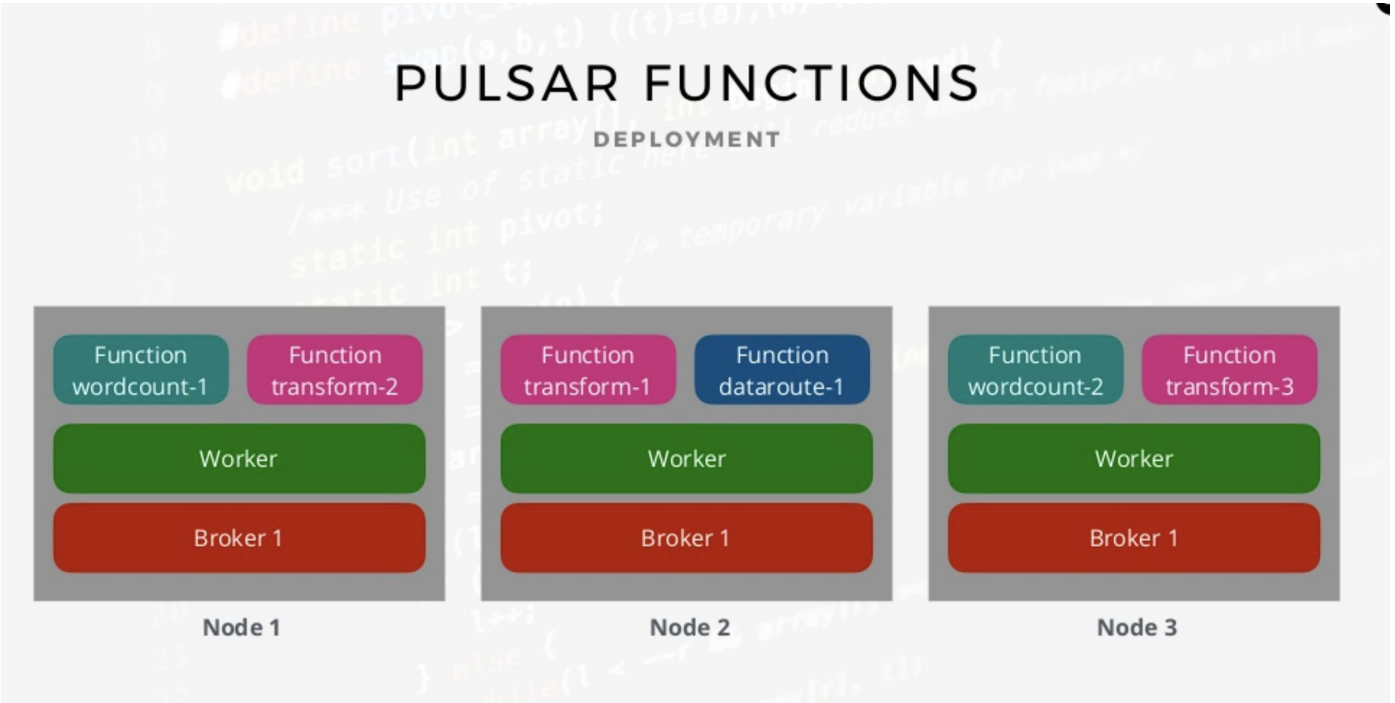
Pulsar Functions:- A Brief Introduction

Function lifecycle

Flexible execution environments

- Pulsar managed
  - Thread
  - Process
- Externally managed
  - Kubernetes

CRUD based Rest API



Pulsar Functions:- A Deep Dive

Agenda

Brief introduction to Pulsar Functions

Deep Dive into internals

- Submission workflow
- Scheduling workflow
- Execution workflow
- Java Instance concepts

Current/Future Work

Pulsar Functions:- Submission Workflow

Function Representation

Submit to any worker

Json repr of FunctionConfig

- tenant/namespace/name
- Input/Output
- configs
- lot more knobs ….

Function Code

- jars/.py/zip/etc

FunctionConfig

```
public class FunctionConfig {
    private String tenant;
    private String namespace;
    private String name;
    private String className;
    private Collection<String> inputs;
    private String output;
    private ProcessingGuarantees processingGuarantees;
    private Map<String, Object> userConfig;
    private Map<String, Object> secrets;
    private Integer parallelism;
    private Resources resources;
    ...
}
```

Pulsar Functions:- Submission Workflow

Submission Checks

AuthN/AuthZ checks

FunctionConfig validation

- missing parameters
- Incorrect parameters
- Local Configs

Function Code Validation

- class presence, etc

Copy Code to Bookeeper

FunctionMetaData

```
message FunctionMetaData {
    FunctionDetails functionDetails;
    PackageLocationMetaData packageLocation;
    uint64 version;
    uint64 createTime;
    map<int32, FunctionState> instanceStates;
    FunctionAuthenticationSpec functionAuthSpec;
}
```

Pulsar Functions:- Submission Workflow

Function MetaData Manager

System of record

Stores all Functions

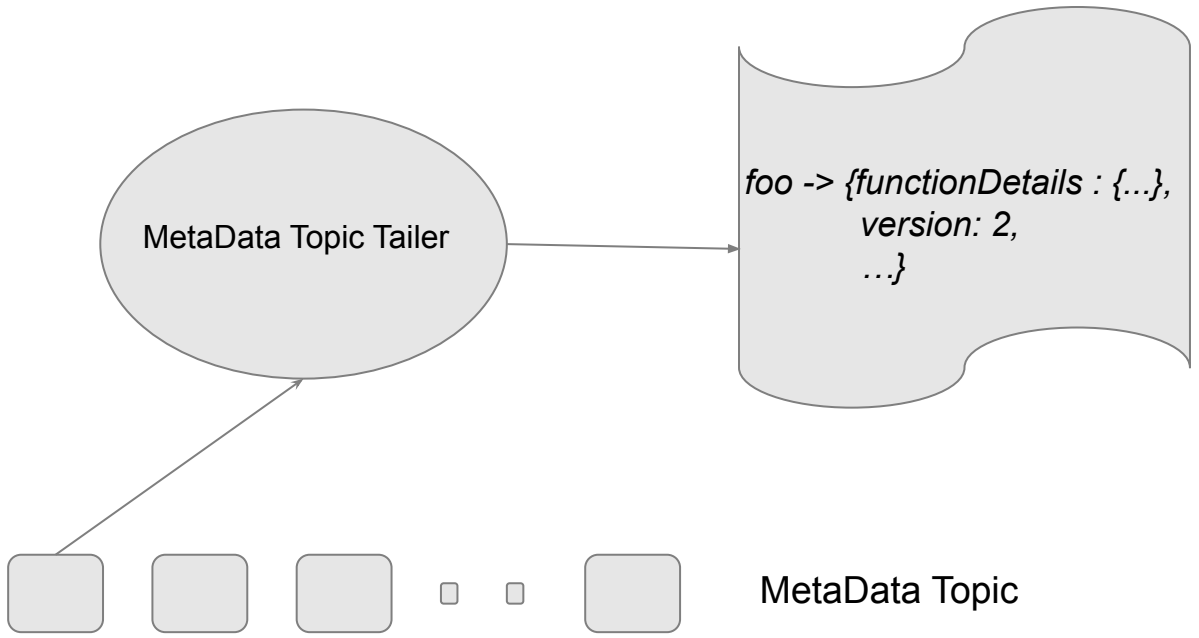
- map from <FQFN, FunctionMetaData>

FQFN:- Fully Qualified Function Name

Backed by Pulsar Topic

- Function MetaData Topic

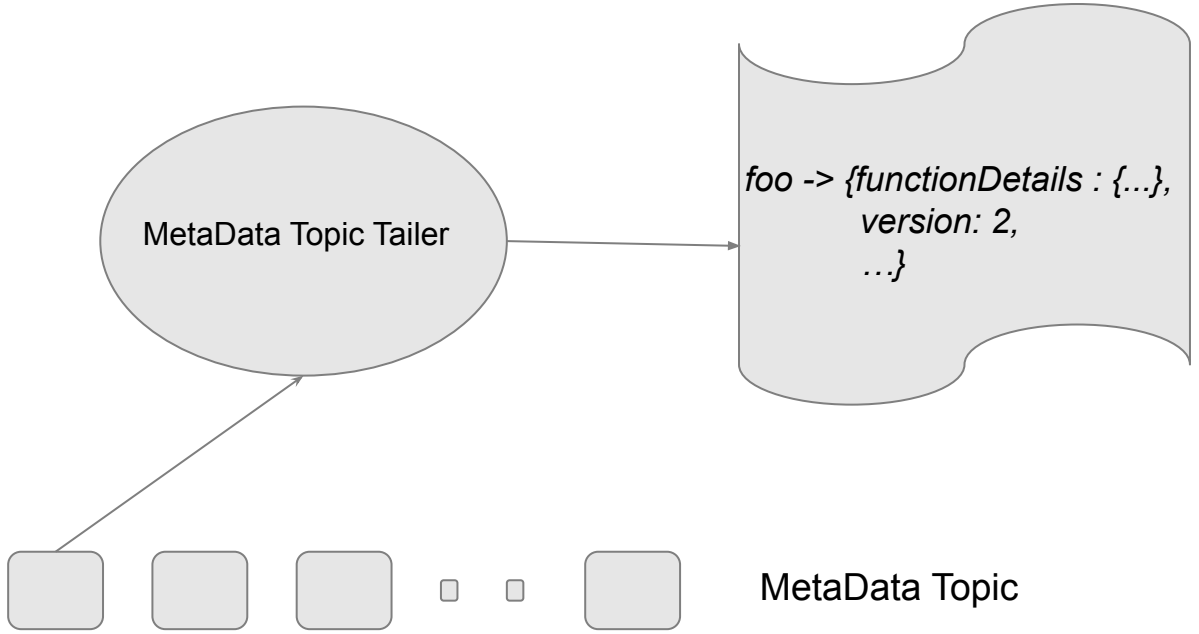
Contains a MetaData Topic Tailer



Pulsar Functions:- Submission Workflow

Function MetaData Manager:- Update State Machine

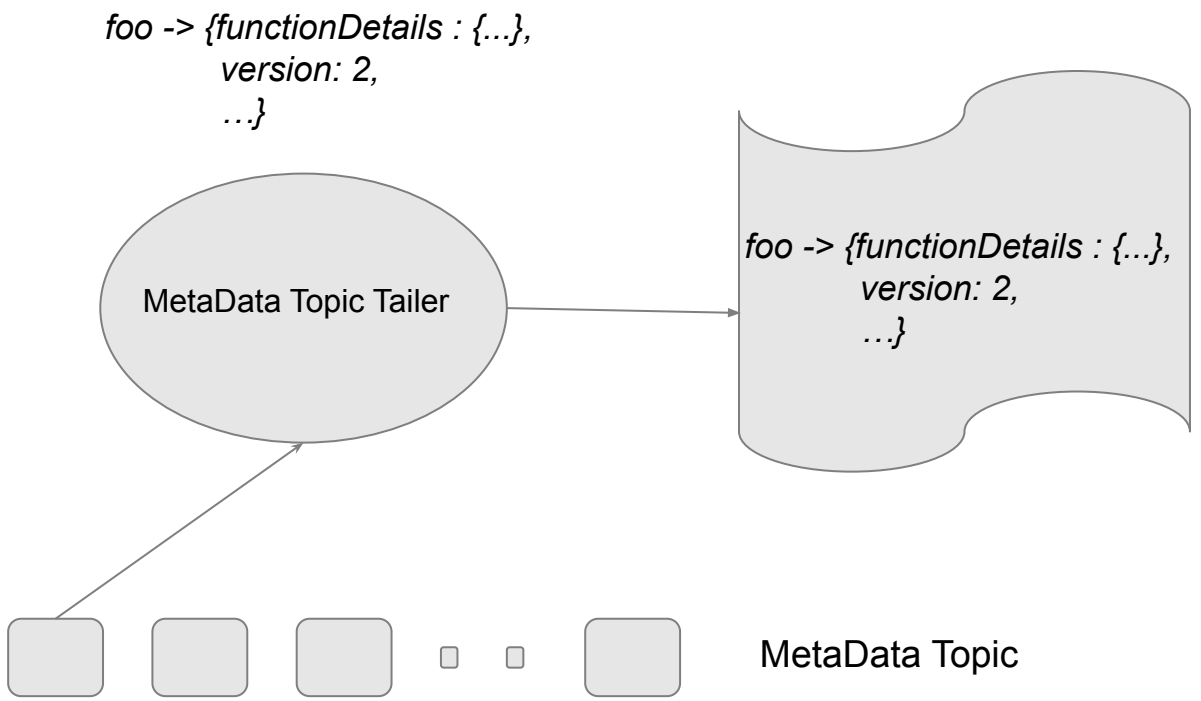
Just before Function creation/update/delete



Pulsar Functions:- Submission Workflow

Function MetaData Manager:- Update State Machine

Make a copy of the current state

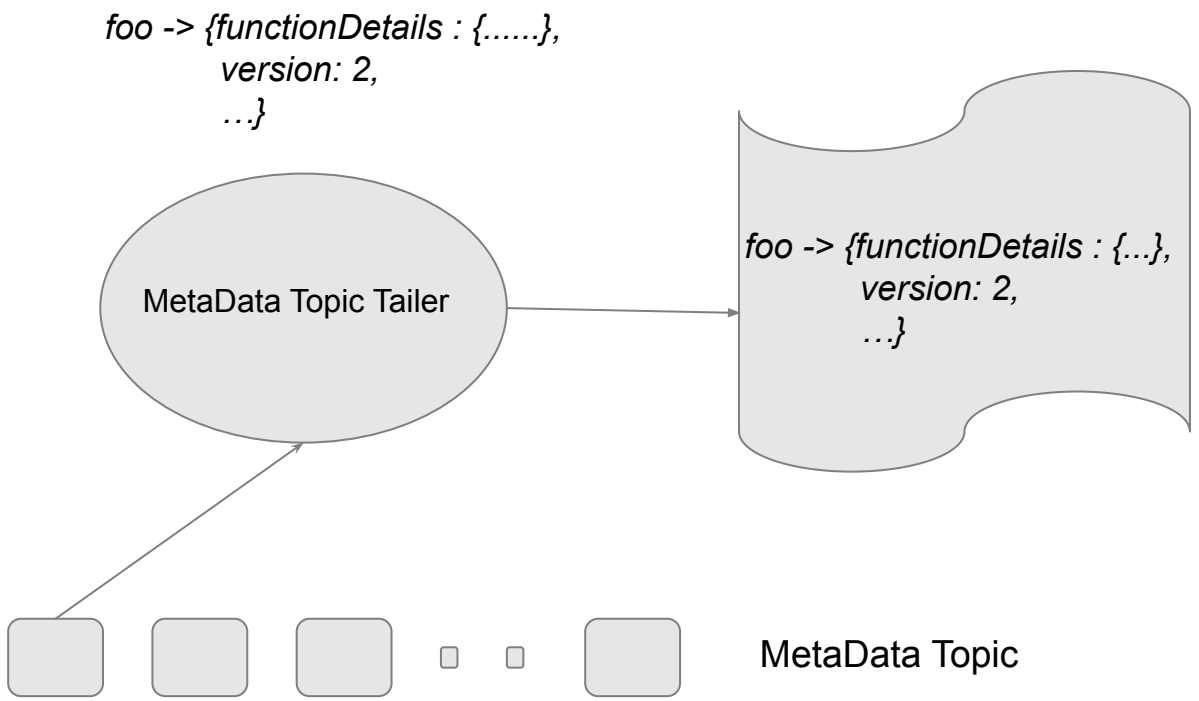


Pulsar Functions:- Submission Workflow

Function MetaData Manager:- Update State Machine

Make a copy of the current state

Merge the updates



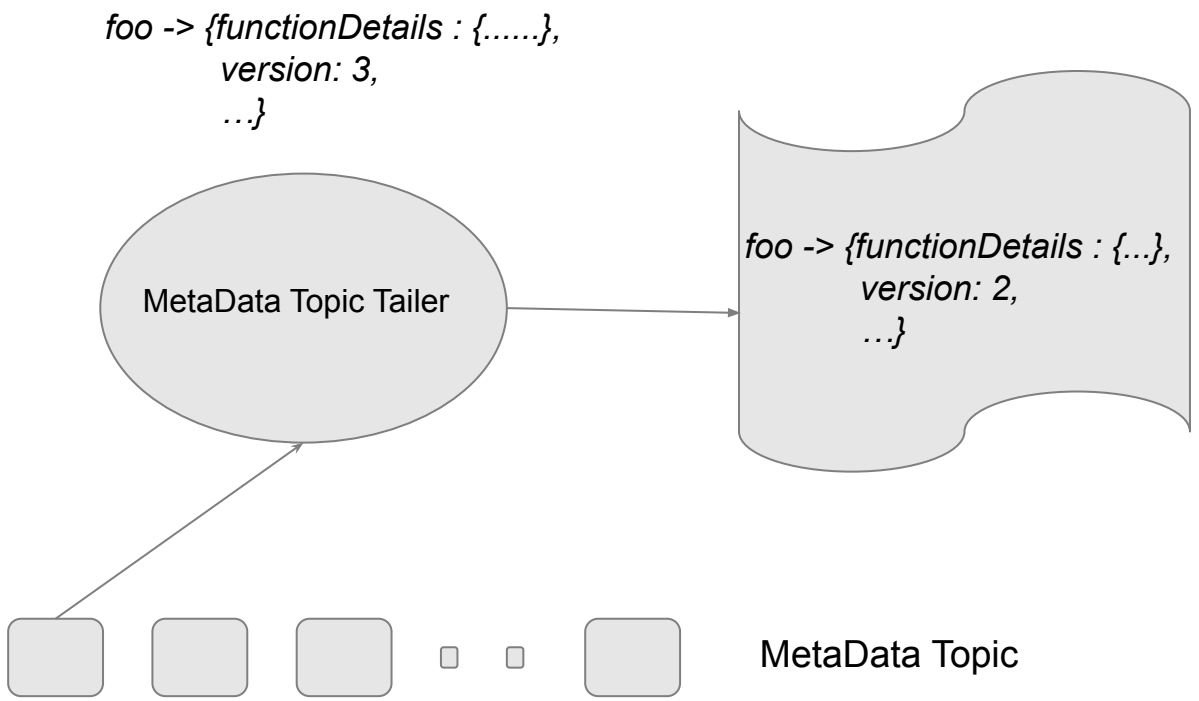
Pulsar Functions:- Submission Workflow

Function MetaData Manager:- Update State Machine

Make a copy of the current state

Merge the updates

Increment the version



Pulsar Functions:- Submission Workflow

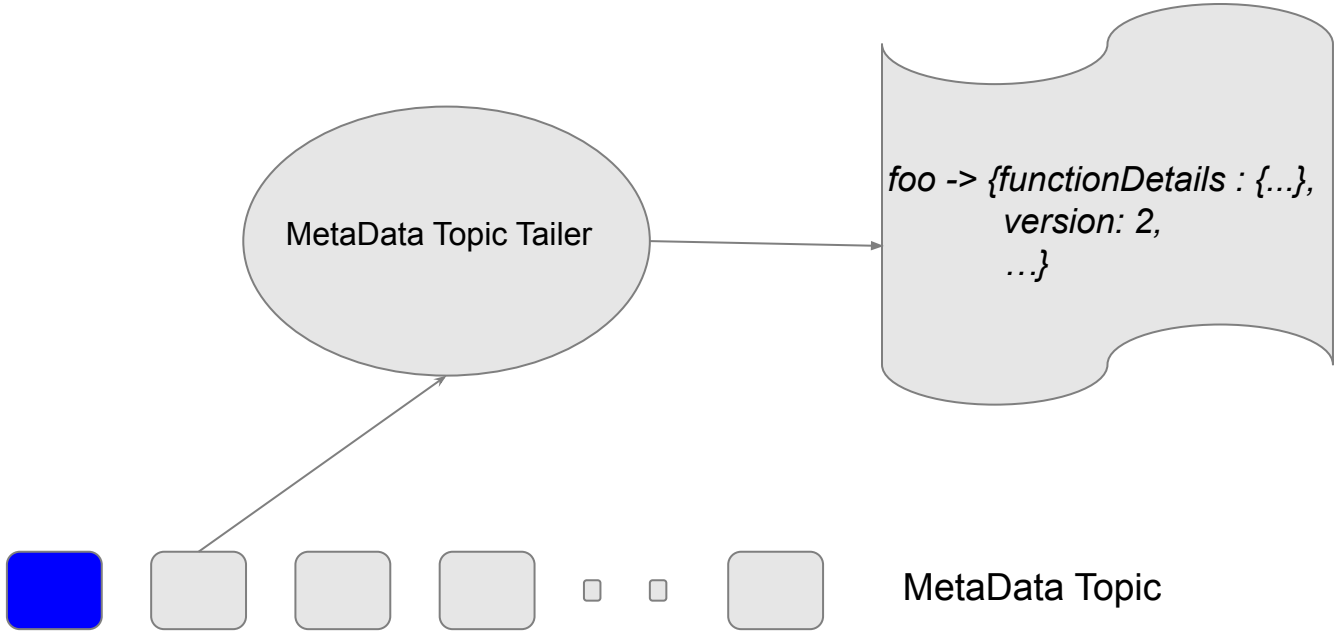
Function MetaData Manager:- Update State Machine

Make a copy of the current state

Merge the updates

Increment the version

Write to MetaData Topic



Pulsar Functions:- Submission Workflow

Function MetaData Manager:- Update State Machine

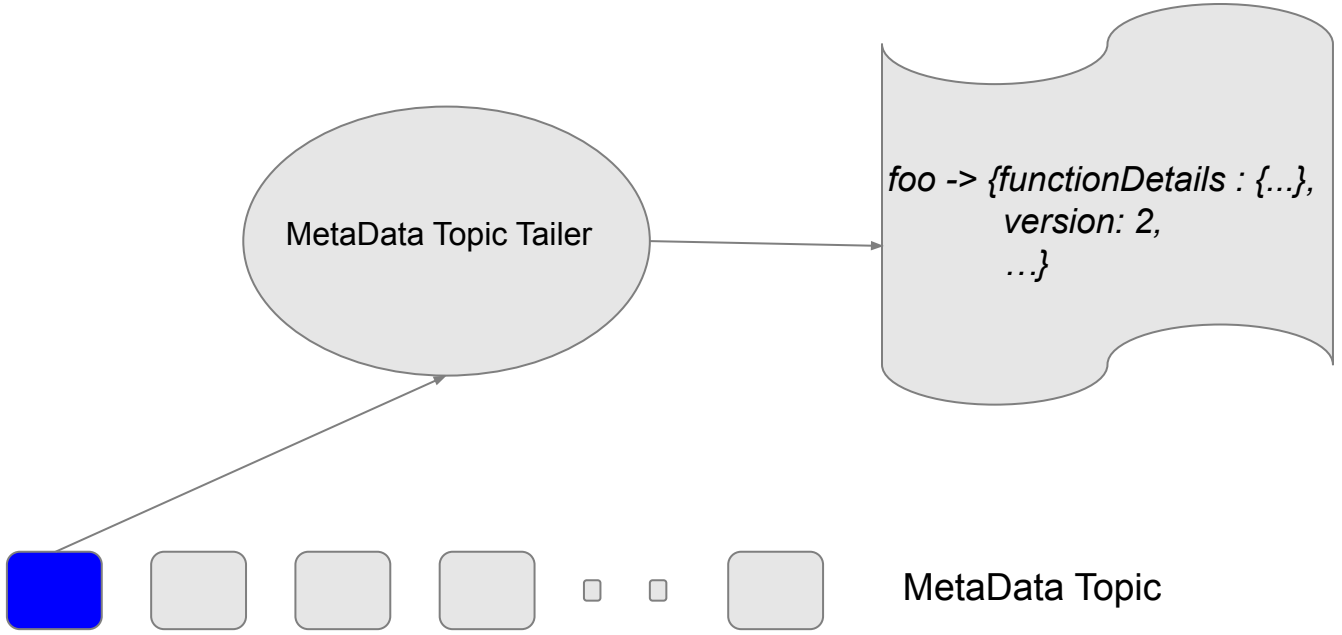
Make a copy of the current state

Merge the updates

Increment the version

Write to MetaData Topic

Tailer reads and verifies



Pulsar Functions:- Submission Workflow

Function MetaData Manager:- Update State Machine

Make a copy of the current state

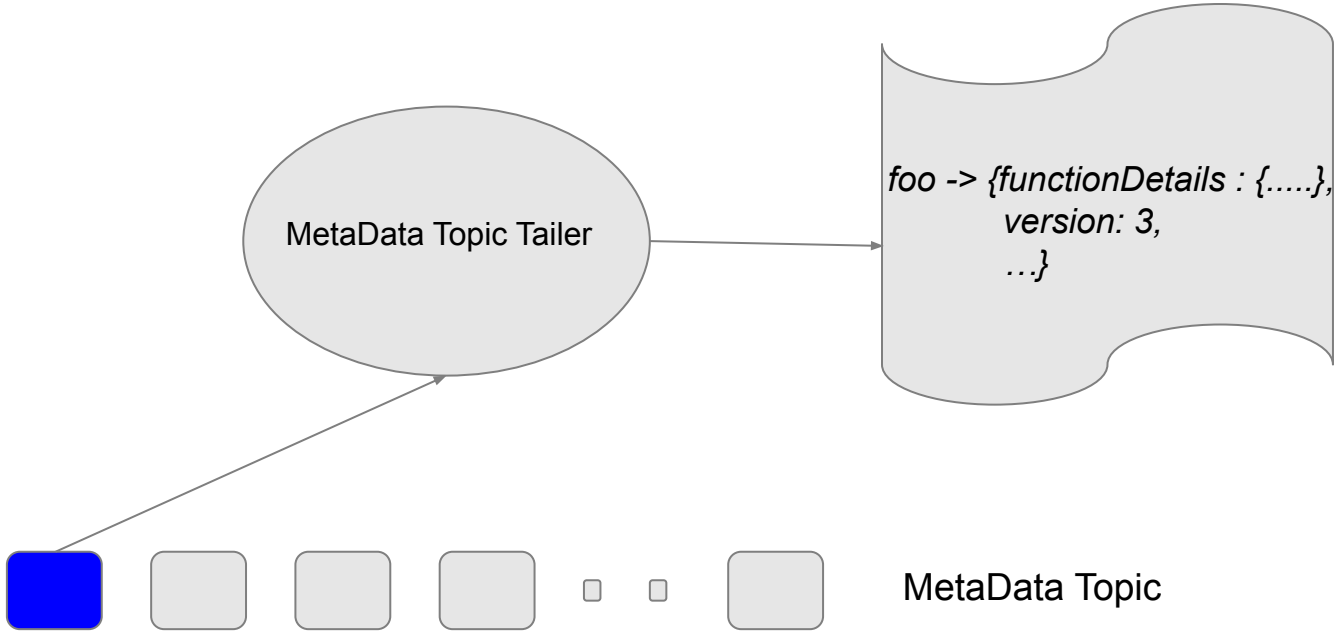
Merge the updates

Increment the version

Write to MetaData Topic

Tailer reads and verifies

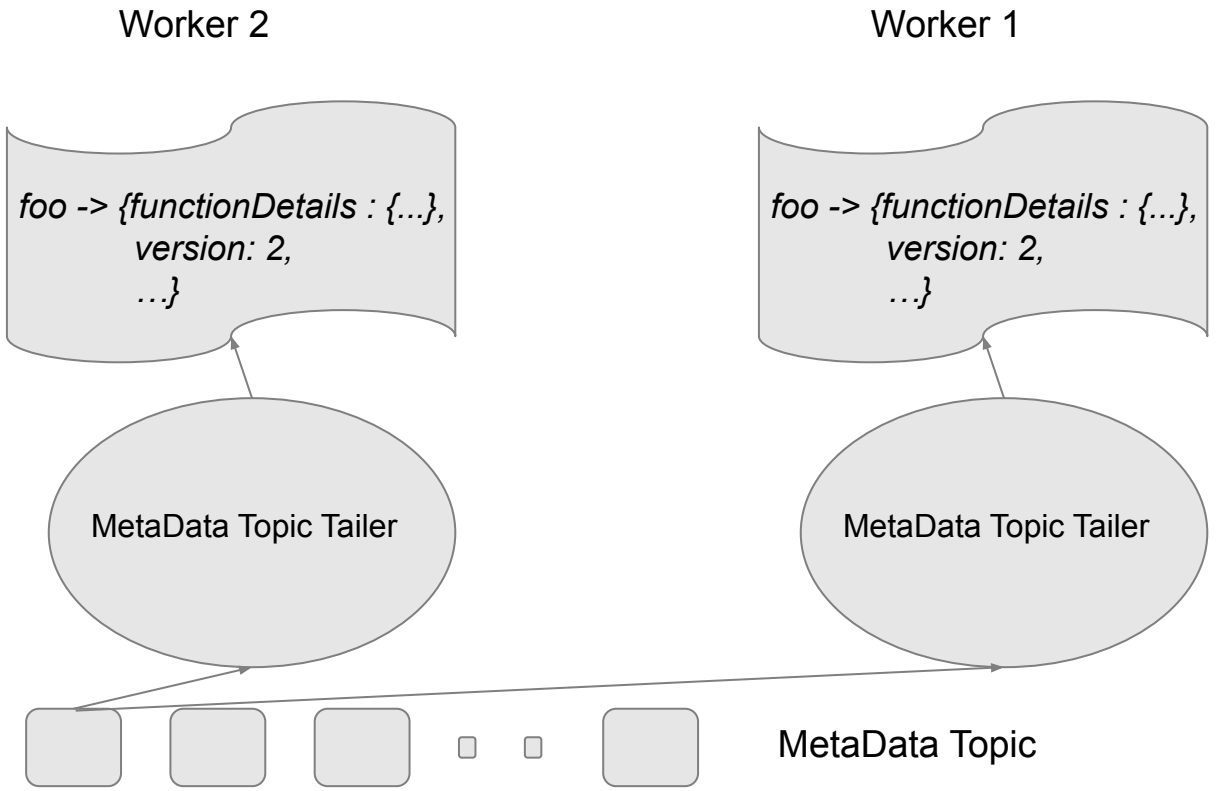
Upon no conflict, tailer updates



Pulsar Functions:- Submission Workflow

Function MetaData Manager:- When do conflicts occur?

Multiple Workers

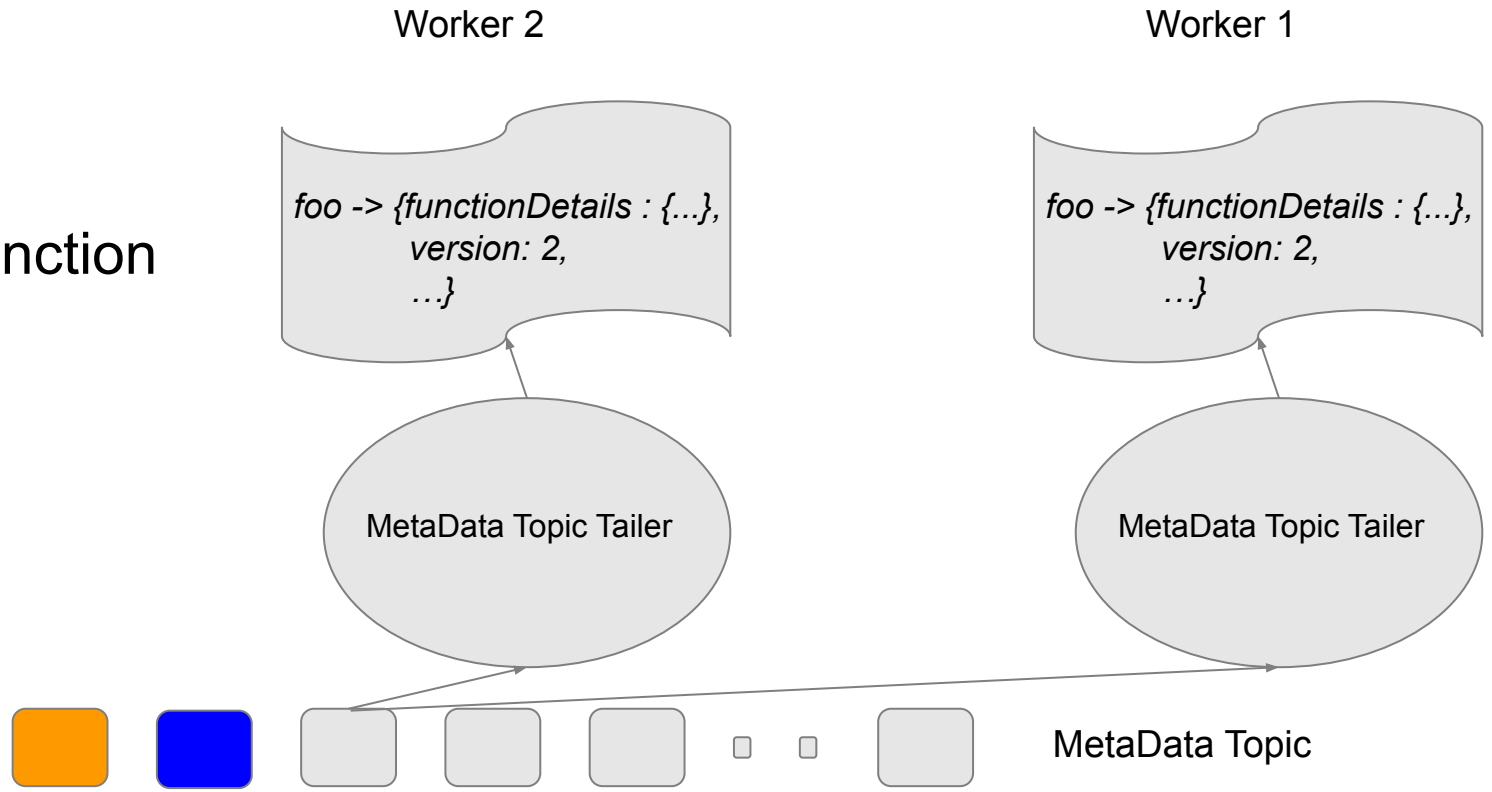


Pulsar Functions:- Submission Workflow

Function MetaData Manager:- When do conflicts occur?

Multiple Workers

Concurrent updates to same function



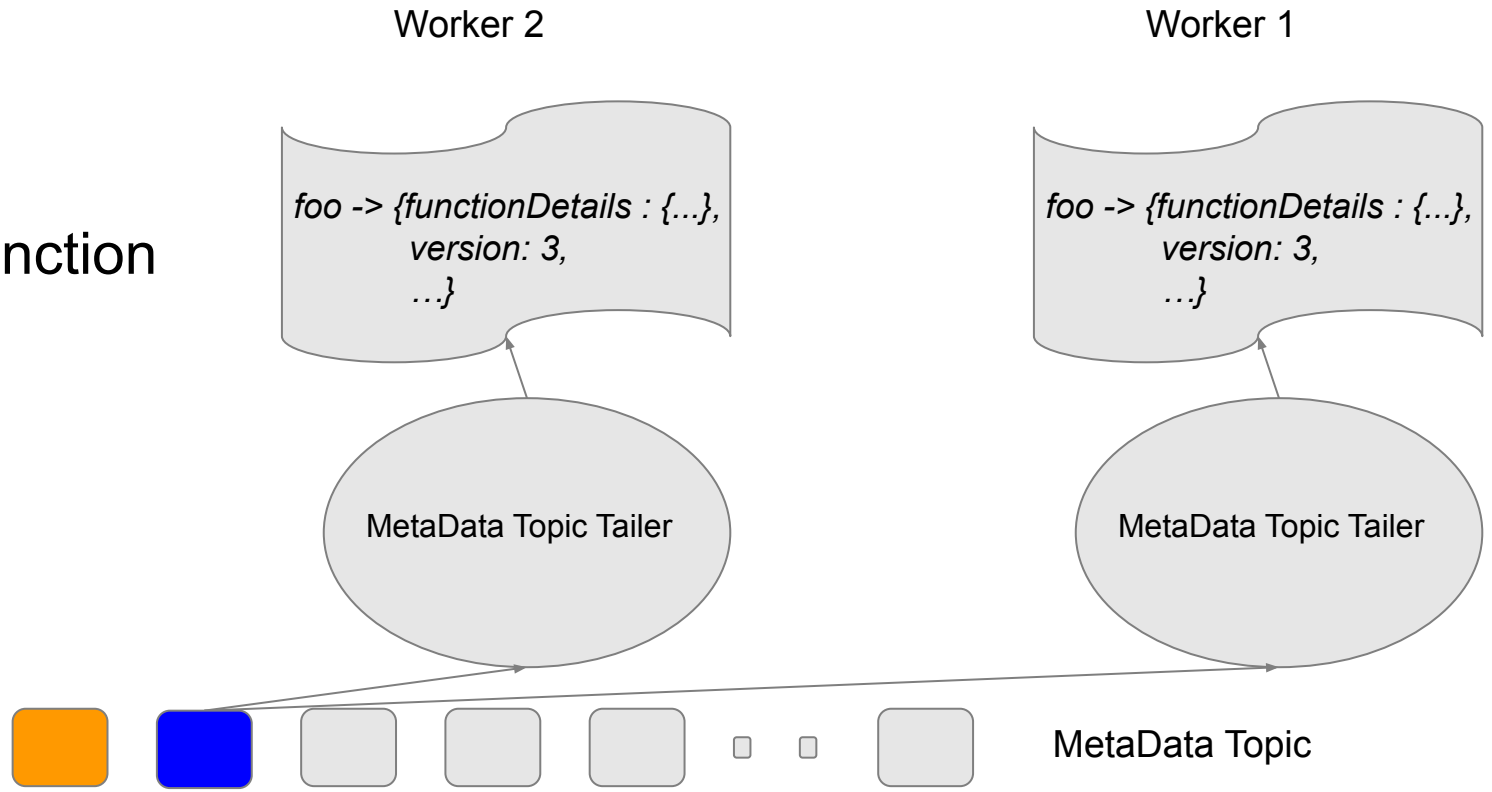
Pulsar Functions:- Submission Workflow

Function MetaData Manager:- When do conflicts occur?

Multiple Workers

Concurrent updates to same function

First Writer Wins



Pulsar Functions:- Submission Workflow

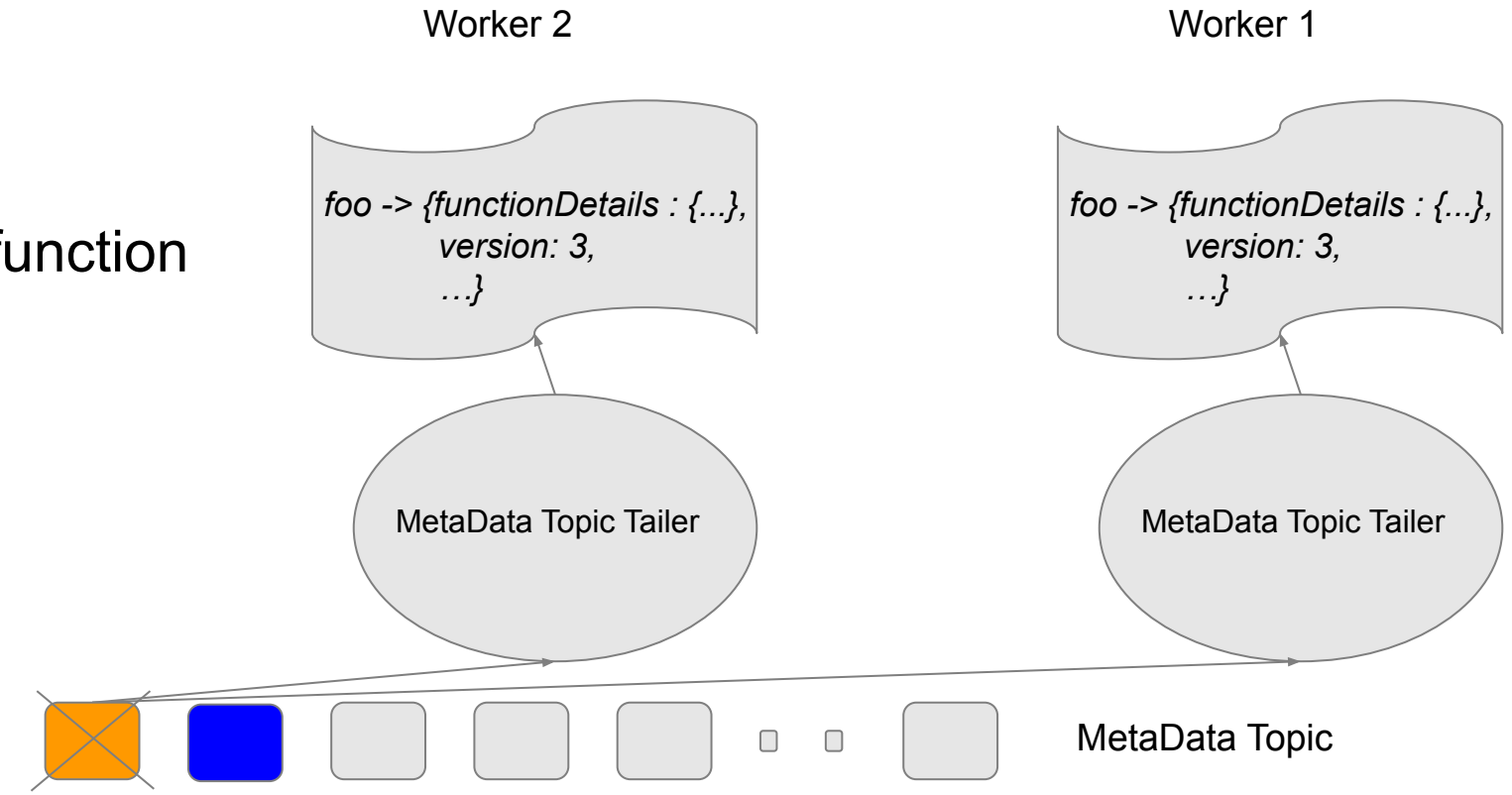
Function MetaData Manager:- When do conflicts occur?

Multiple Workers

Concurrent updates to same function

First Writer Wins

Others are rejected



Pulsar Functions:- Submission Workflow

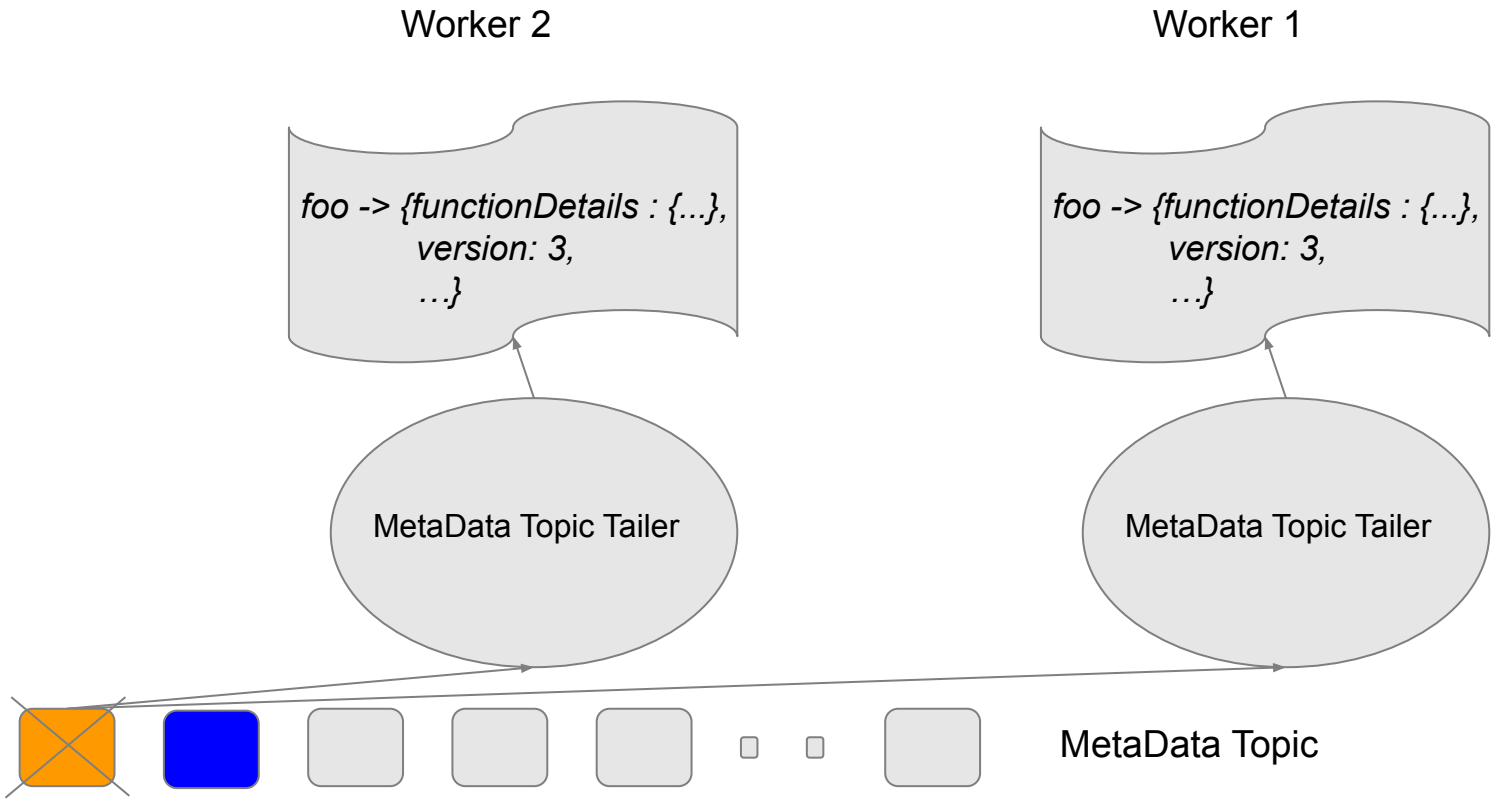
Advantages

Submit to any worker

Validation load scales linearly

Deterministic State Machine

MetaData Topic is audit log



Pulsar Functions:- Submission Workflow

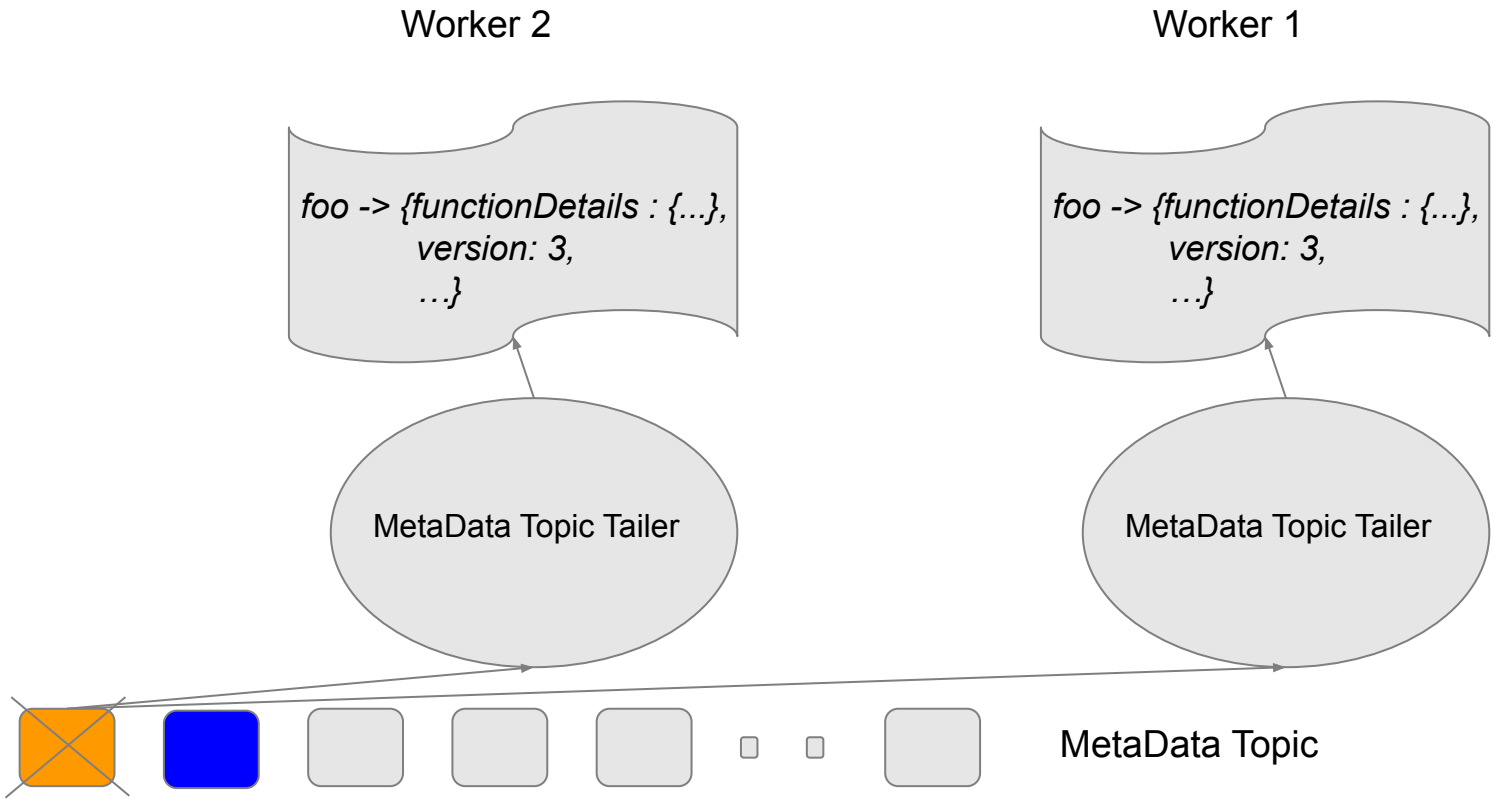
Pitfalls

MetaData topic topic growth

MetaData Topic compaction non-trivial

Worker Start time

All Workers know everything



Pulsar Functions:- A Deep Dive

Agenda

Brief introduction to Pulsar Functions

Deep Dive into internals

- Submission workflow
- Scheduling workflow
- Execution workflow
- Java Instance concepts

Current/Future Work

Pulsar Functions:- Scheduling Workflow

Pluggable Scheduler

Abstracts out Scheduler

Executed only on a Leader

Invoked when

- Function CRUD operations
  - create/update
  - delete
- Worker Changes
  - Unresponsive/dead workers
  - New workers
  - Periodic
  - Leadership changes

IScheduler Interface

```
public interface IScheduler {
    List<Assignment> schedule(<List<Instance> unassigned,
                              List<Instance> current,
                              Set<String> workers);
}
```

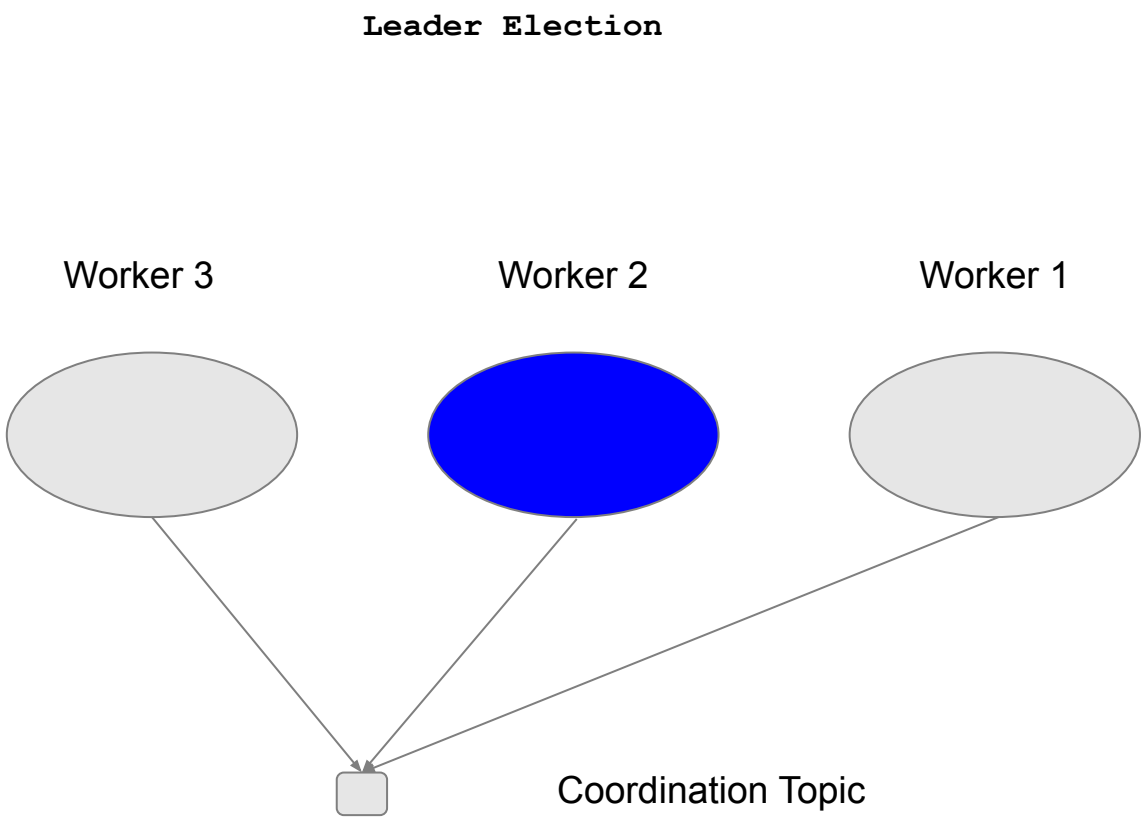
Pulsar Functions:- Scheduling Workflow

Leader Election

Empty Coordination Topic

Failover Subscription

Active Consumer is the Leader



Pulsar Functions:- Scheduling Workflow

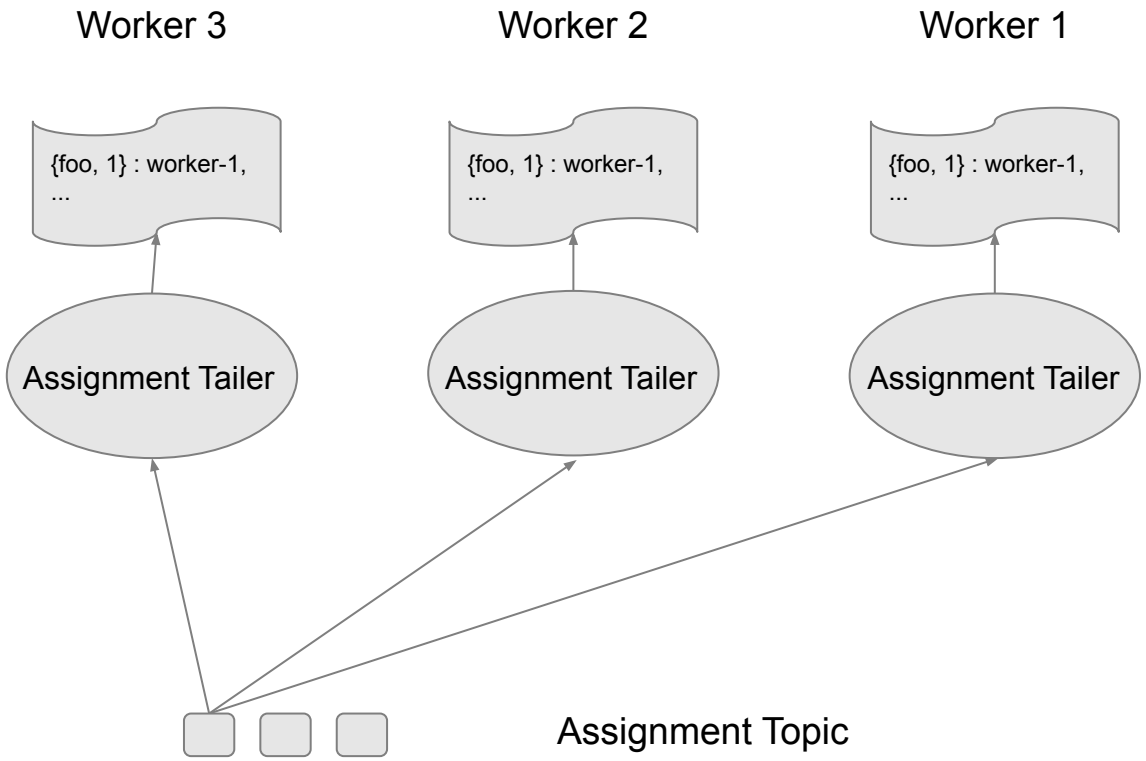
Function Assignments

Assignment Topic

Written by the Leader

Compacted based on key(FQFN + Instance Id)

All workers know about all assignments



Pulsar Functions:- Scheduling Workflow

Assignment Topic

Stores Assignment

Compacted

Key -> (FQFN + InstanceId)

Assignment

```
message Instance {
     FunctionMetaData functionMetaData = 1;
     int32 instanceId = 2;
}

message Assignment {
     Instance instance = 1;
     string workerId = 2;
}
```

Pulsar Functions:- A Deep Dive

Agenda

Brief introduction to Pulsar Functions

Deep Dive into internals

- Submission workflow
- Scheduling workflow
- Execution workflow
- Java Instance concepts

Current/Future Work

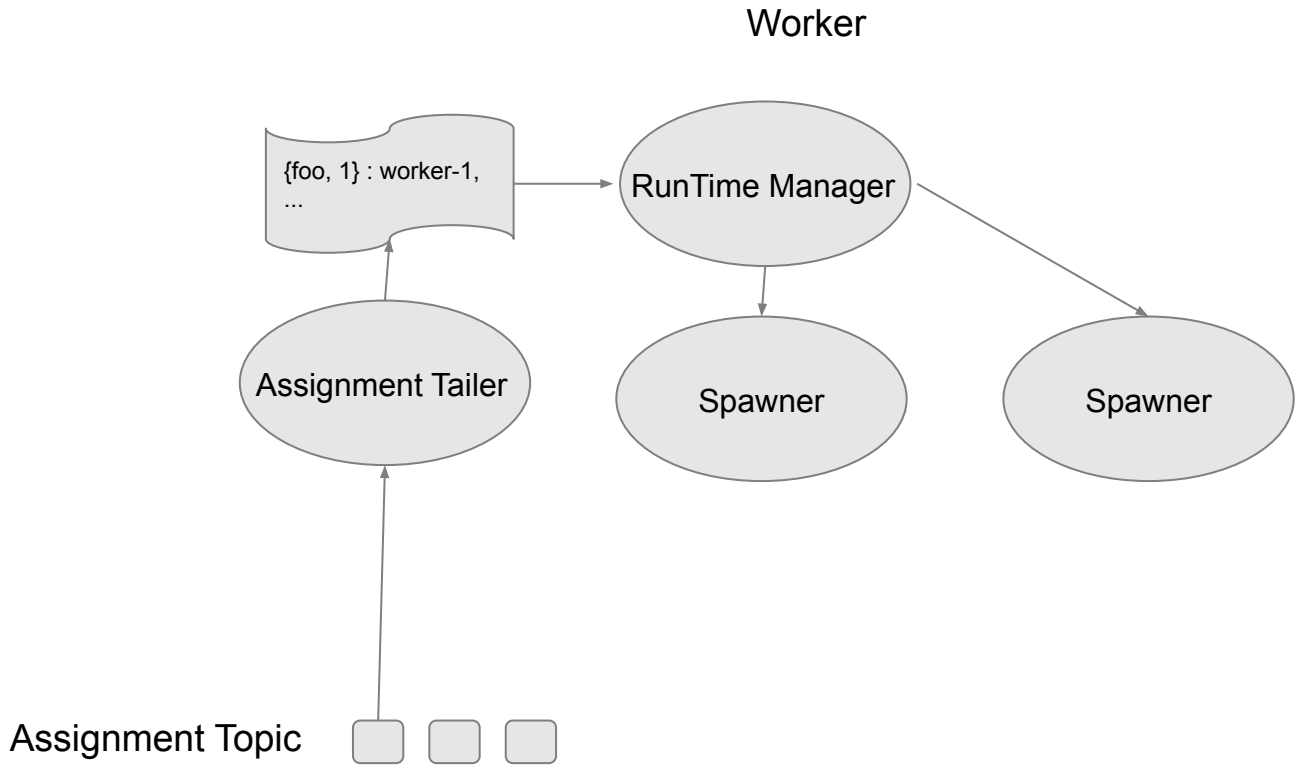
Pulsar Functions:- Execution Workflow

Function RunTime Manager

Triggered by Changes to Assignment Table

Takes care of the worker's specific assignments

Function lifecycle management via Spawner



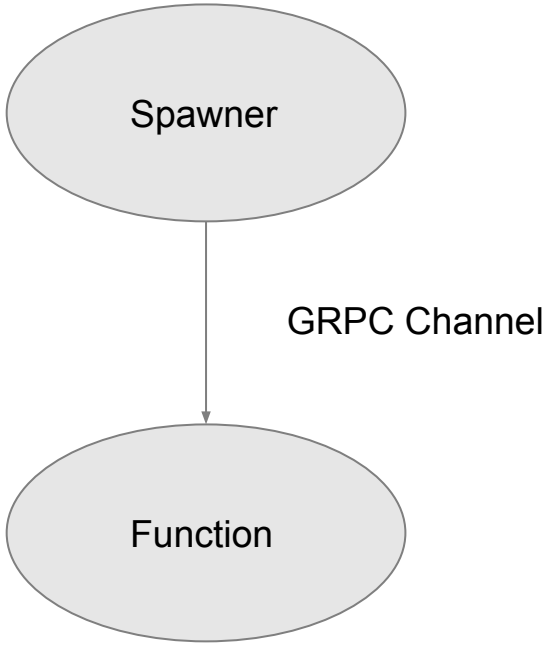
Pulsar Functions:- Execution Workflow

Spawner

Abstracts out execution environments using Runtime Factory

Manages Function lifecycle

Maintains grpc connection with Function instance

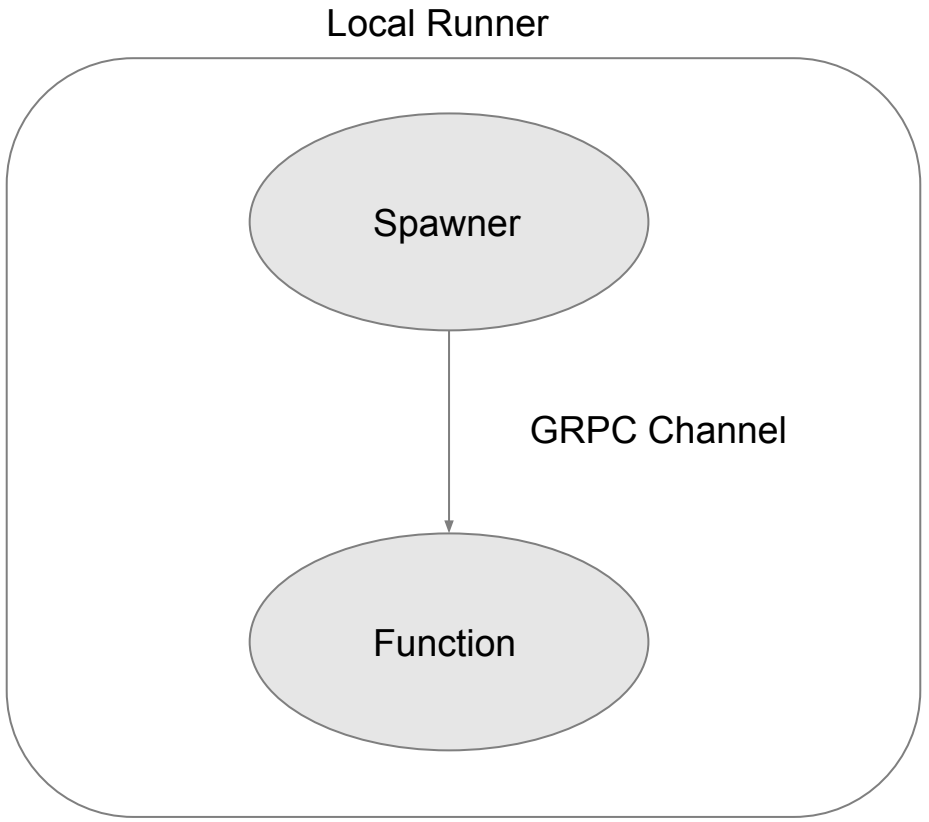


Pulsar Functions:- Execution Workflow

Local Runner

Short-circuit MetaData Manager and Runtime Manager

Directly use Spawner



Pulsar Functions:- Execution Workflow

Runtime Factory

Simple interface for creating execution environments

Creates Runtimes

Runtime Factory

```
public interface RuntimeFactory {

    void initialize(WorkerConfig workerConfig);

    Runtime createContainer(InstanceConfig instanceConfig,
                            String codeFile);

    void close();
}
```

Pulsar Functions:- A Deep Dive

Agenda

Brief introduction to Pulsar Functions

Deep Dive into internals

- Submission workflow
- Scheduling workflow
- Execution workflow
- Java Instance concepts

Current/Future Work

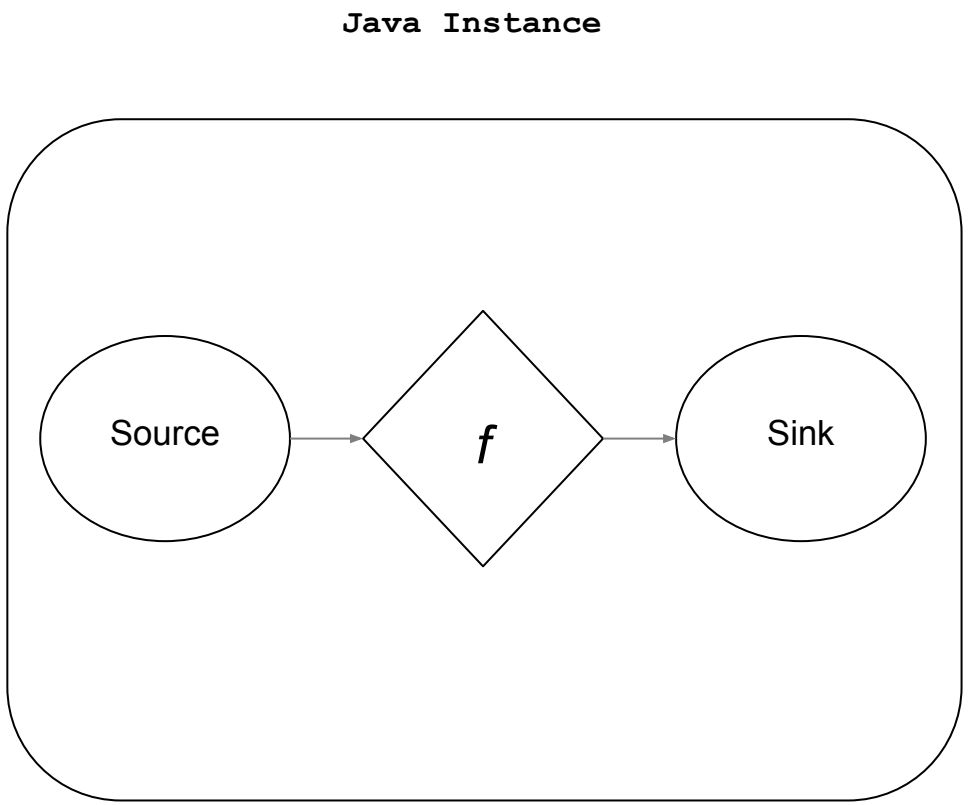
Pulsar Functions:- Java Instance

Source -> Process -> Sink

Java Instance is (source, function, sink) ensemble.

Source abstracts reading from input topics

Sink abstracts writing to output topic

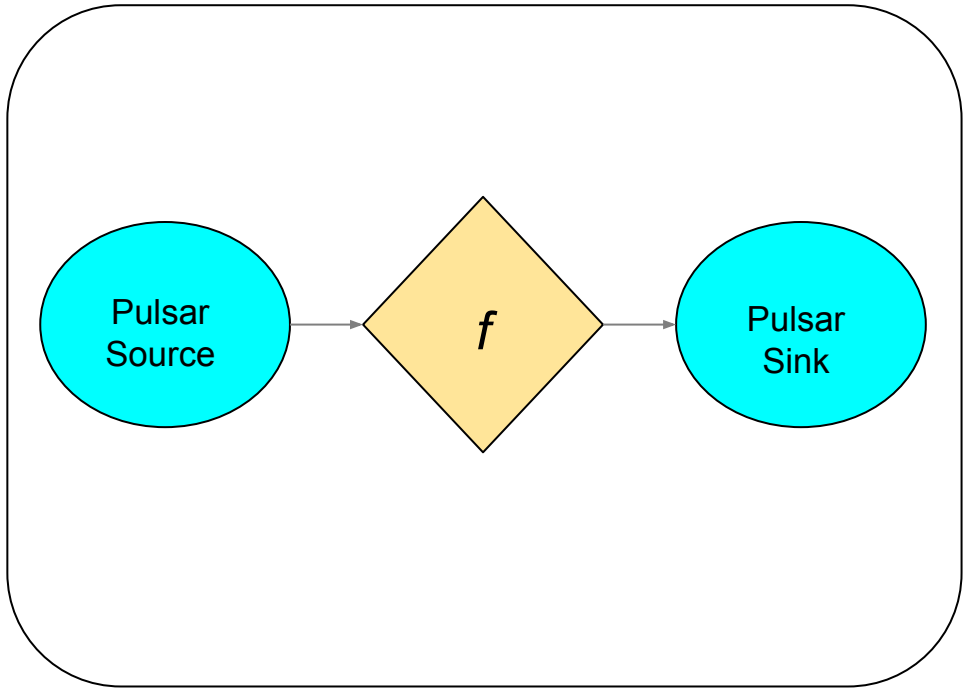


Pulsar Functions:- Java Instance

Regular Pulsar Functions

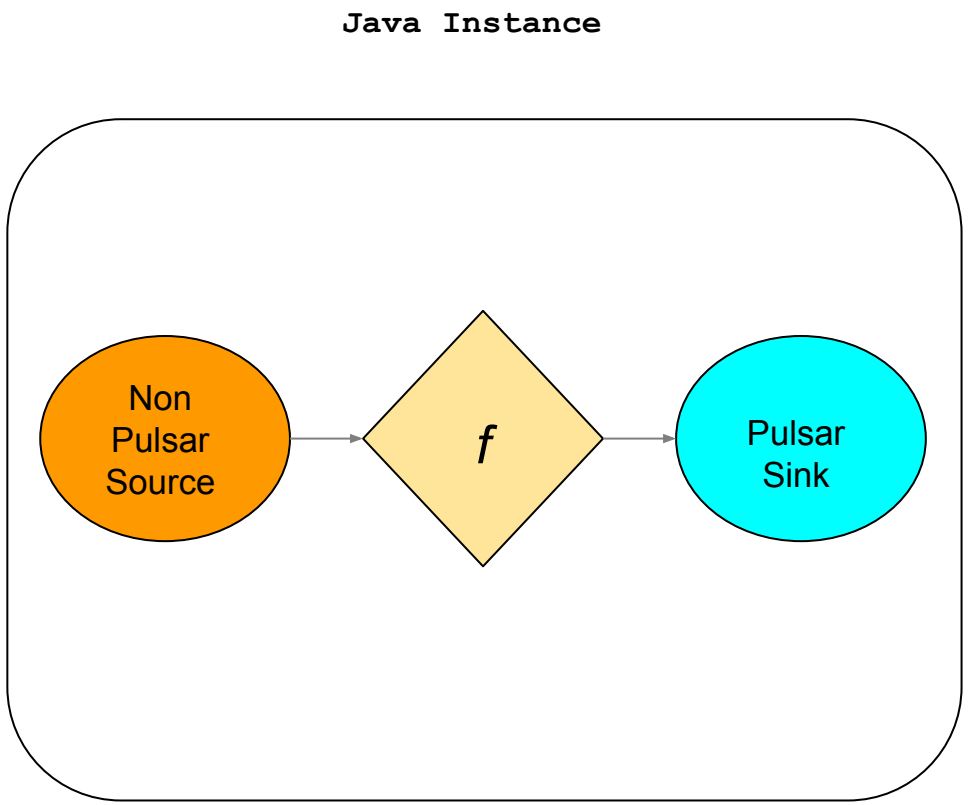
Pulsar Source implements the Source interface to read from Pulsar topics

Pulsar Sink implements Sink interface to write to Pulsar topic



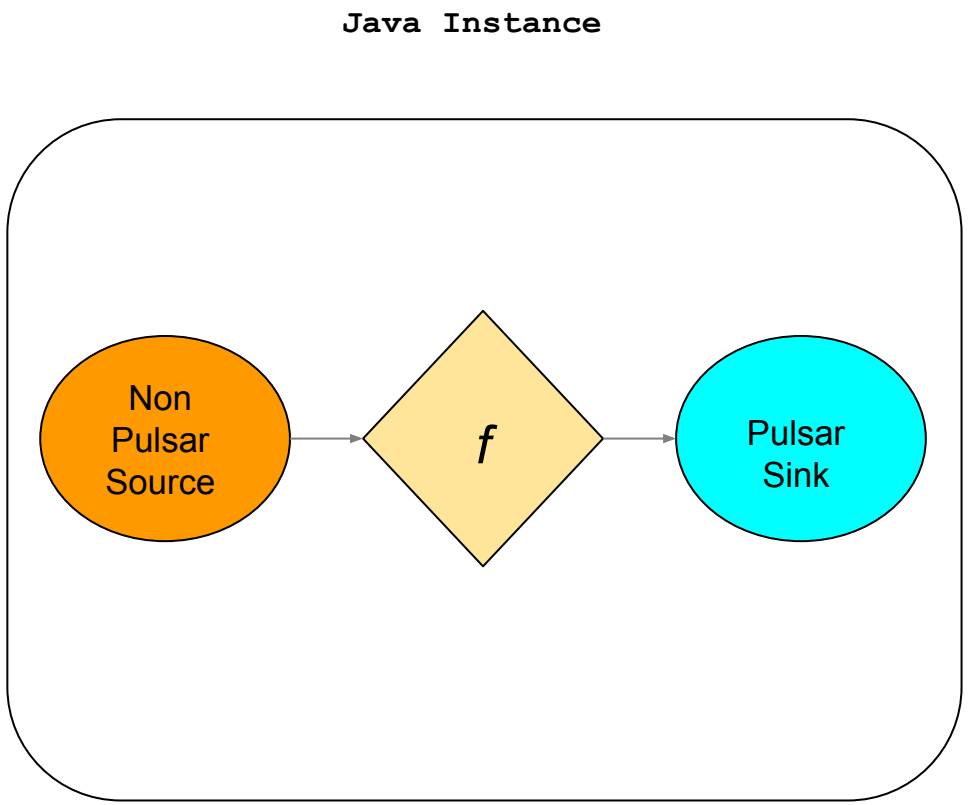
Pulsar Functions:- Java Instance

What if we have non-Pulsar Source?



Pulsar Functions:- Java Instance

Pulsar IO



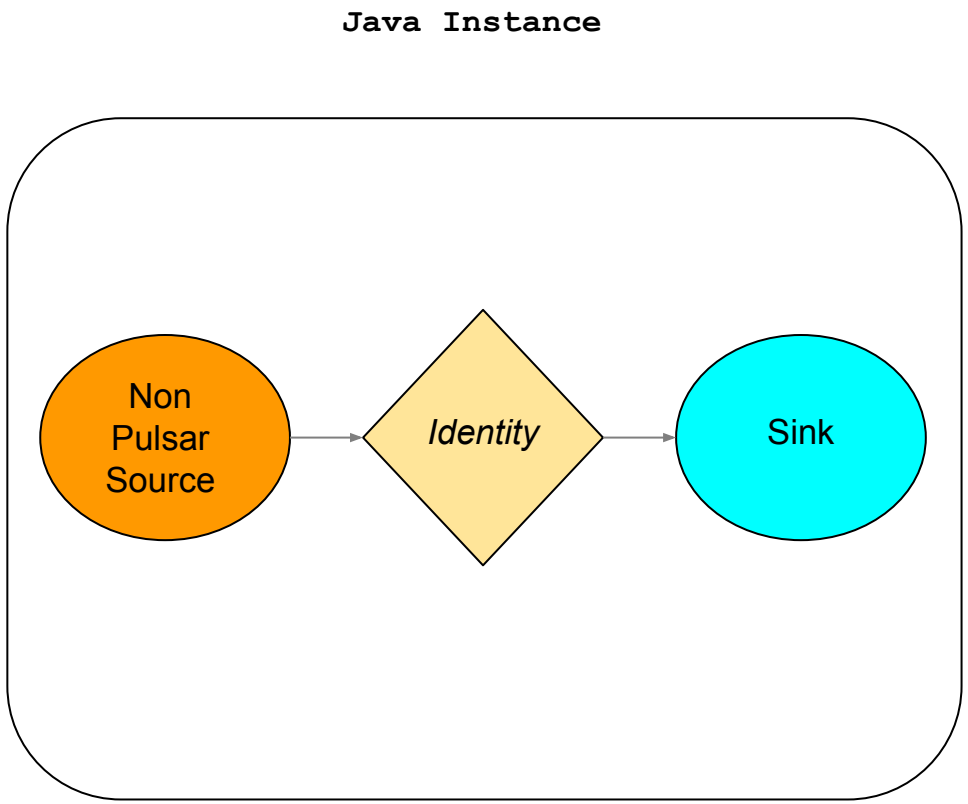
Pulsar Functions:- Java Instance

Pulsar IO Source

Non Pulsar Source reads from external system

Identity Function lets the data pass thru

Pulsar Sink writes to Pulsar



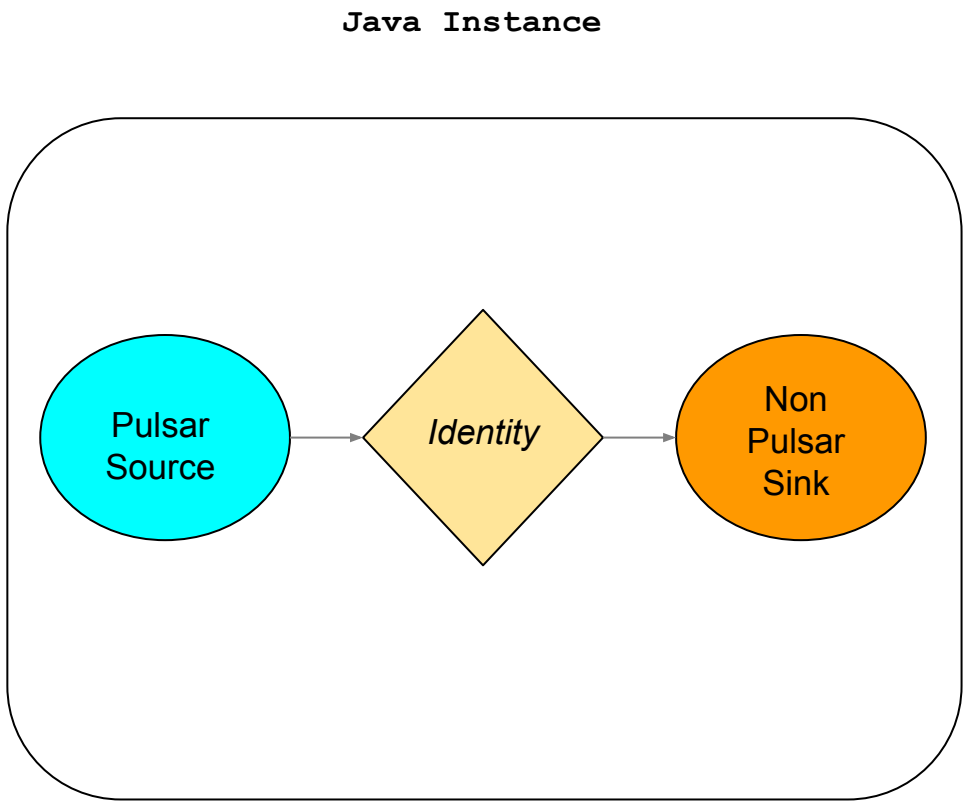
Pulsar Functions:- Java Instance

Pulsar IO Sink

Pulsar Source reads from Pulsar topics

Identity Function lets the data pass thru

Non Pulsar Sink writes to an external system



Pulsar Functions:- A Deep Dive

Agenda

Brief introduction to Pulsar Functions

Deep Dive into internals

- Submission workflow
- Scheduling workflow
- Execution workflow
- Java Instance concepts

Current/Future Work

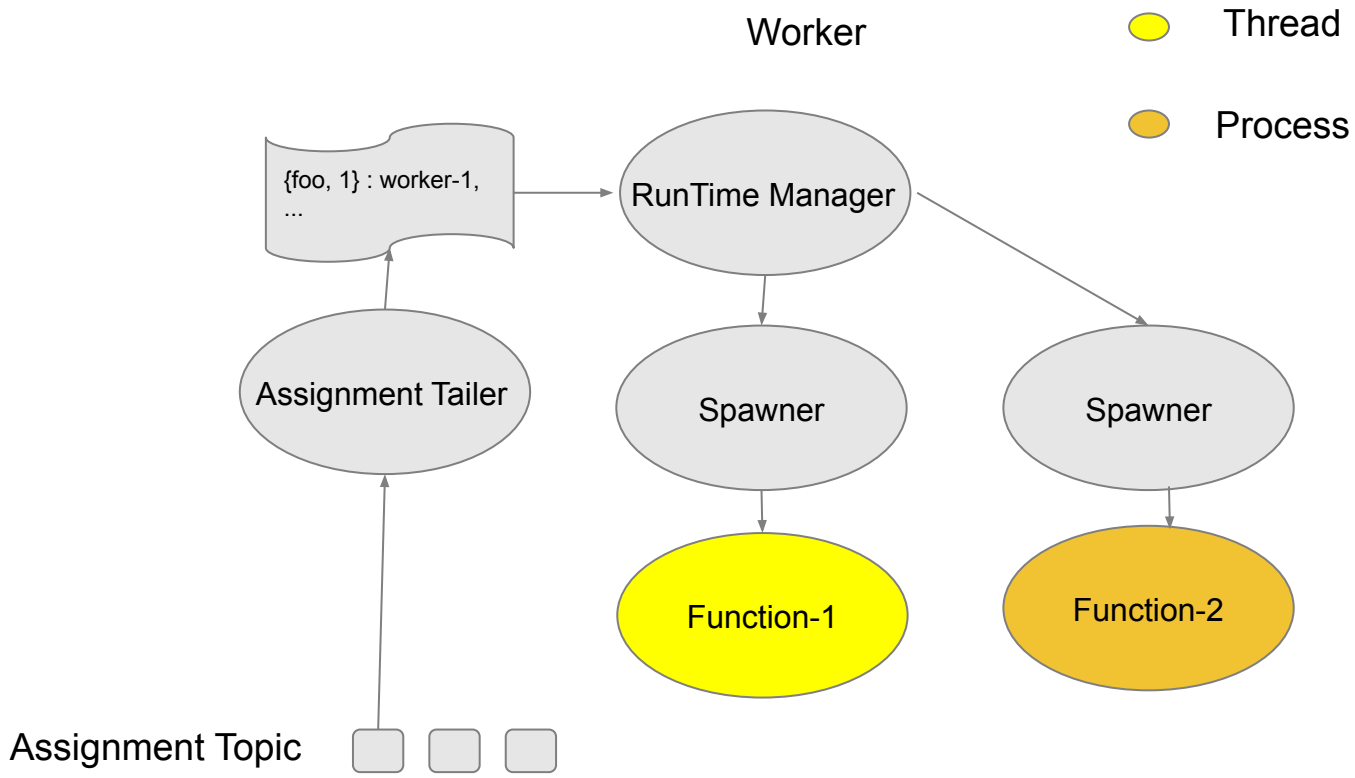
Pulsar Functions:- Future Work

Dynamic Runtime Selection

Each setup only supports a static Runtime(Process/Thread/Pods)

Change it to be dynamically specified during submission

Function RunTime Manager Changes



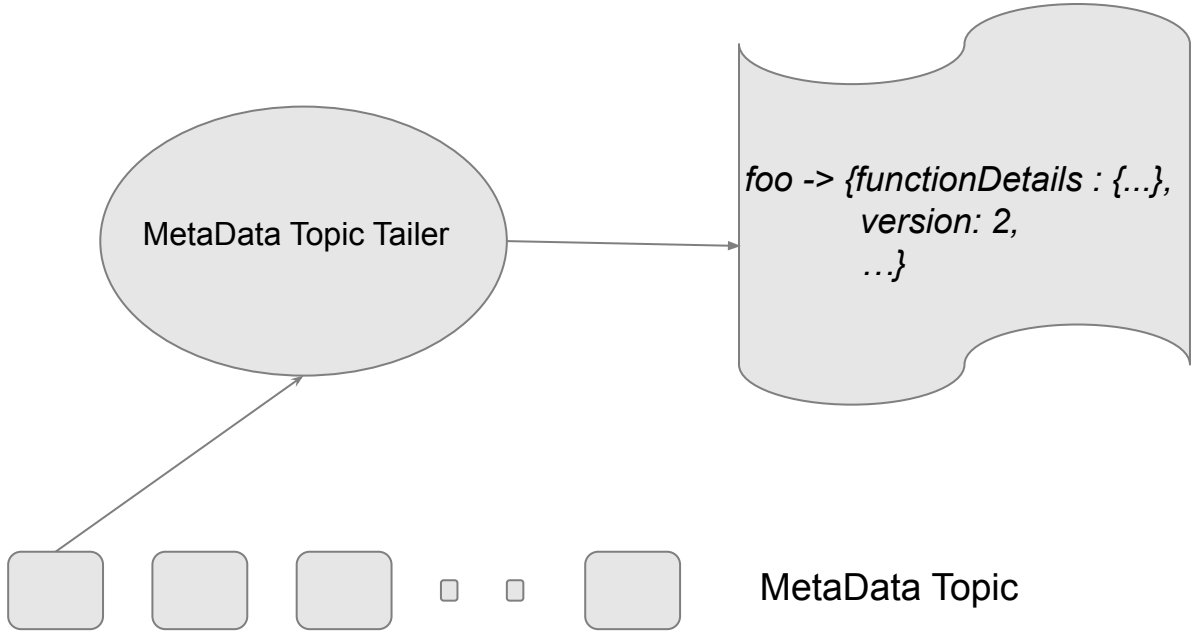
Pulsar Functions:- Future Work

Function MetaData Topic Compaction

MetaData Topic not compacted

Stores all function change requests

Worker needs to read from beginning upon startup



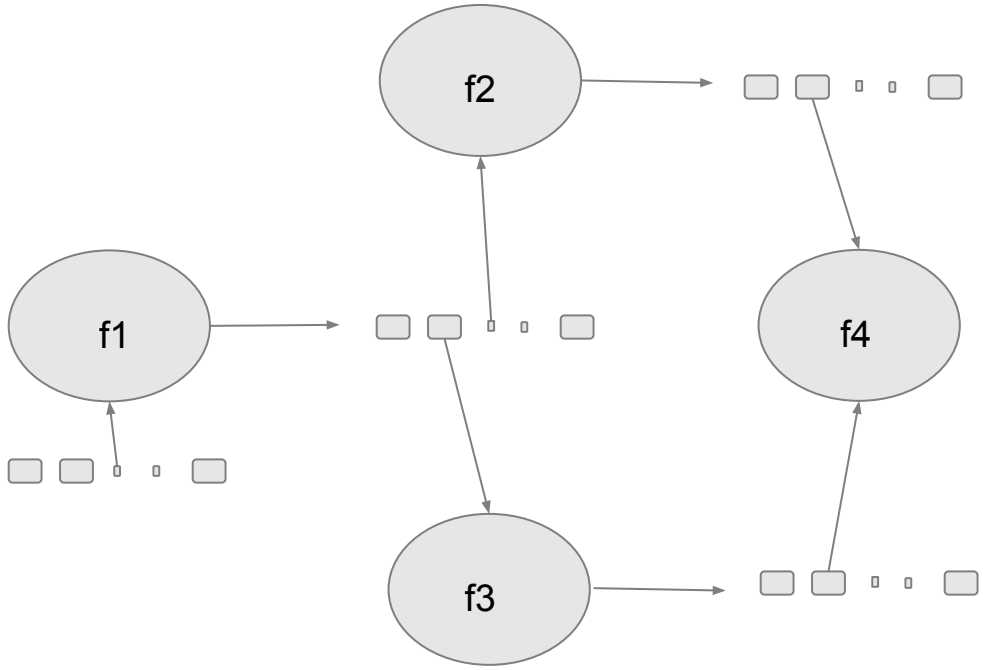
Pulsar Functions:- Future Work

Function Mesh

Chaining Functions

Output of one going as input of others

A simple workflow API



Pulsar Functions:- Future Work

Batch Sources

Discover/Collect Cycle

Repeating Cycle

Don’t drop discovered tasks on failures

BatchSource

```
public interface BatchSource<T> {

    void open(final Map<String, Object> config,
              SourceContext context);

    void discover(Consumer<byte[]> taskEater);

    void prepare(byte[] task);

    Record<T> readNext();
}
```

Thank You

splunk> turn data into doing™